hitb magazine

KEEPING KNOWLEDGE FREE

Volume 1, Issue 6, June 2011 *magazine.hackinthebox.org*



BOTNET Resistant Coding 24

Windows NUMERIC HANDLE Allocation In Depth 48

Cover Story

SOCIAL SECURITY 42

Advertisement

Google™



A Place To Be You

Chances are you have a good idea of where you want to go in life. At Google, we've designed a culture that helps you get there.

We're hiring!

Apply online: www.google.com/EngineeringEMEA

© 2010 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc.

hitb magazine

Volume 1, Issue 6, June 2011

Editorial

Hello readers and welcome to the summer release of Issue 06!

We've got loads of awesome content lined up as always including a feature article/interview with Joe Sullivan, Chief Security Officer at social network behemoth Facebook and keynoter at the 2nd annual HITBSecConf in Europe. Along side Joe, we also sat down with Chris Evans who participated in the keynote panel discussion on the Economics of Vulnerabilities to talk about Google's Vulnerability Rewards program.

While we're on the subject of our 2nd annual HITBSecConf, HITB2011AMS, the .MY and .NL teams did a fantastic job as always with over 45 international speakers joining us for 2 days of madness! We had some pretty kick ass presentations including a special live EMV (EuroPay MasterCard Visa) hack and the much sought after 'ELEVATOR' from Stefan 'i0nic' Esser. Read on for the special report in this issue from our friends at Random Data (a Hackerspace in Utrecht) who not only participated in the Hackerspaces Village but also won the ITQ Hackerspaces Challenge featuring Lego Mindstorms! Photos from the event are up on *http://photos.hitb.org/*

June also sees us celebrating the next phase in the (r)evolution of the HITB news portal with the launch of the all new HITB landing page and HITBSecNews site (http://news.hitb.org). Powered by Drupal 7.2, the portal features a slick new layout with full social media integration so you can now link your Facebook or Twitter accounts when commenting on stories or sharing articles.

Enjoy the zine, have a great summer and get your spy glasses ready for Issue 007's special feature on spy/surveillance gadgets!

Dhillon "L33tdawg" Kannabhiran,
Founder/Chief Executive Officer, Hack in The Box



Editor-in-Chief
Zarul Shahrin
http://twitter.com/zarulshahrin

Editorial Advisor
Dhillon Andrew Kannabhiran

Technical Advisor
Matthew "j00ru" Jurczyk

Design
Shamik Kundu
(*cognitive.designs@gmail.com*)

Website
Bina Kundu

HITB Magazine – Keeping Knowledge Free
http://magazine.hackinthebox.org

Contents

EVENTS
HITB 2011 Amsterdam 04
Random Data Gets In The Box 10

WEB SECURITY
Next Generation Web Attacks – HTML 5, DOM (L3) and XHR (L2) 14



NETWORK SECURITY
Botnet-Resistant Coding 24

LINUX SECURITY
The Story of Jugaad 34



COVER STORY
Social Security 42

WINDOWS SECURITY
Windows Numeric Handle Allocation In Depth 48

APPLICATION SECURITY
Hardening Java Applications with Custom Security Policies 58

PROFESSIONAL DEVELOPMENT
CISSP® Corner 68
Books 70

INTERVIEW
Vulnerability Reward Program 72

HITB AMSTERDAM 2011

The Second Annual HITB Deep-Knowledge Security Conference in Europe



"It's like I'm actually in there!"


Lock picking action at the TOOOL.nl booth


Lock picks - Don't leave home without it!


Joffrey Czarny and Elena Kropochkina from Devoteam Security

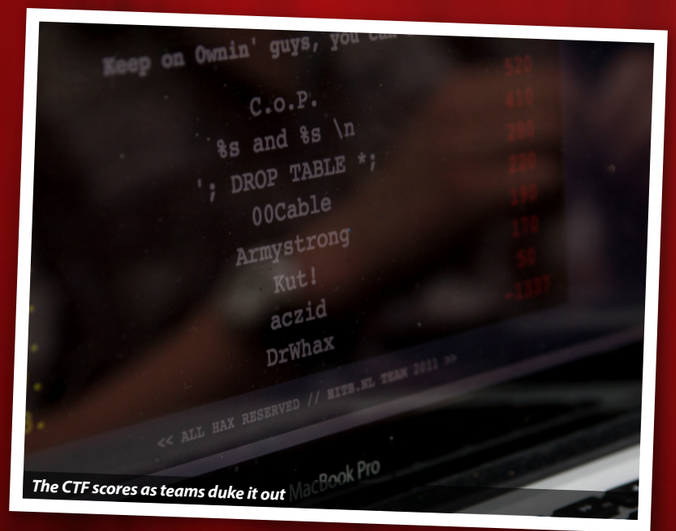

"How much space do we have to pop ES!?"


Twisty knobs


Itzhak 'auk' Avraham - a new speaker to the HITB line up during his talk on ARM / Android exploitation



Stefan 'i0n1c' Esser finally revealed to the whole world what ELEVATOR really is - a joke turned media frenzy!



The CTF scores as teams duke it out


The 2011 edition of the HITB technology showcase area featuring a new Hackerspaces Challenge, Capture The Flag - World Domination competition and of course lock picking village by TOOOL.nl


Don 'The Hunter' Bailey giving AGPS devices the smack down!

Last month, hacker fever once again hit Amsterdam with the 2nd annual HITB security conference in Europe, HITB2011AMS!

Over 45 speakers descended on the NH Grand Krasnapolsky and made it our largest speaker contingent to date with a mind-numbing two days' worth of groundbreaking talks in a quad track format! This year also saw an expanded technology exhibition, an even bigger Hackerspaces Village featured alongside the Capture The Flag - World Domination competition, Lockpicking Village by TOOOL.nl and an all-new addition - The Hackerspaces LEGO Mindstorm Challenge sponsored by ITQ! >

HITB AMSTERDAM 2011


CTF participants 'sweating' under pressure



DOD Cyber Crime Response Team in attendance

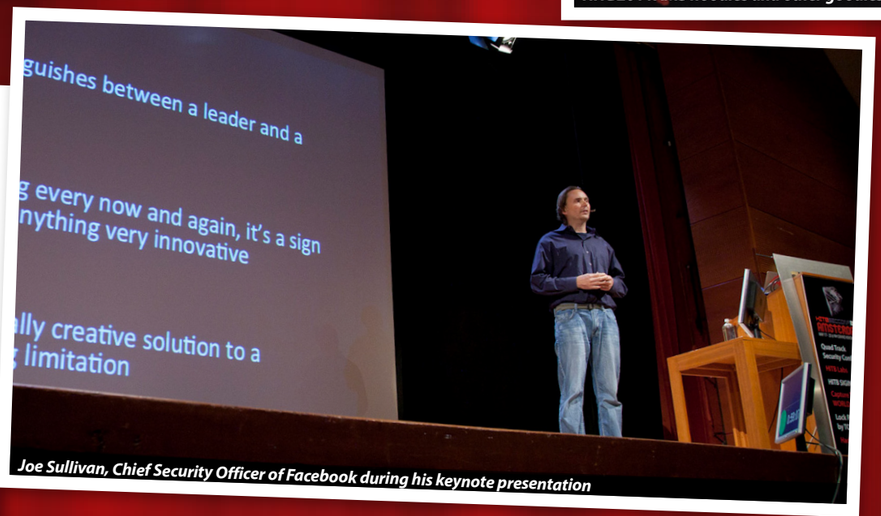

HITB2011AMS hoodies and other goodies



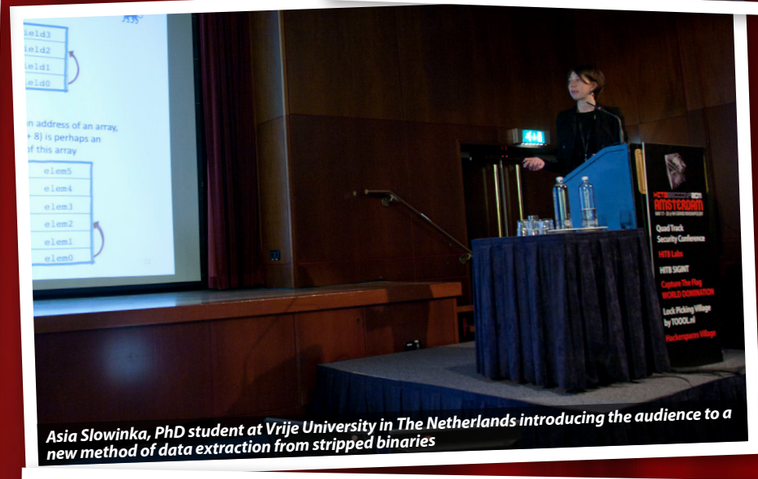
Just Google it!



The Hack42 hackerspace from Arnhem with their retro computing gear



Joe Sullivan, Chief Security Officer of Facebook during his keynote presentation

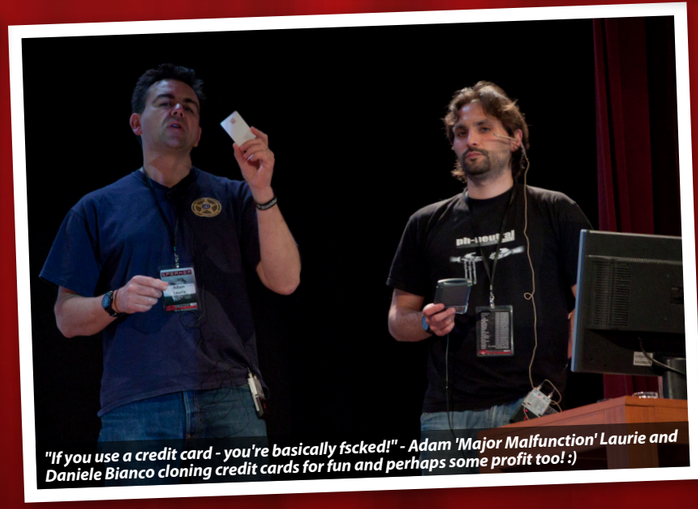

Asia Slowinka, PhD student at Vrije University in The Netherlands introducing the audience to a new method of data extraction from stripped binaries

The quad track conference kicked off with a keynote by Facebook's CSO, Joe Sullivan, followed by a number of killer talks including i0n1c's Antid0te 2.0 - ASLR in iOS presentation where he finally disclosed the details on the much talked about ELEVATOR!

In Day 2's keynote panel on the Economics of Vulnerabilities, Tipping Point, Mozilla, Google, BlackBerry, Adobe and Microsoft fielded some hefty questions from the audience regarding the vulnerability and exploit landscape.

The panel was then followed by further mind bending awesomeness with talks by Raoul Chiesa and Jim Geovedi who were back with ways to make >


Ivan Ristic during his talk on what really breaks SSL


"If you use a credit card - you're basically fscked!" - Adam 'Major Malfunction' Laurie and Daniele Bianco cloning credit cards for fun and perhaps some profit too! :)



Day 2 keynote panel discussion on The Economics of Vulnerabilities featuring (from left): Katie Moussouris (Microsoft), Steve Adegbite (Adobe), Adrian Stone (RIM/BlackBerry), Dhillon '133tdawg' Kannabhiran (HITB/Moderator), Aaron Portnoy (ZDI / TippingPoint), Lucas Adamski (Mozilla) and Chris Evans (Google)

HITB AMSTERDAM 2011



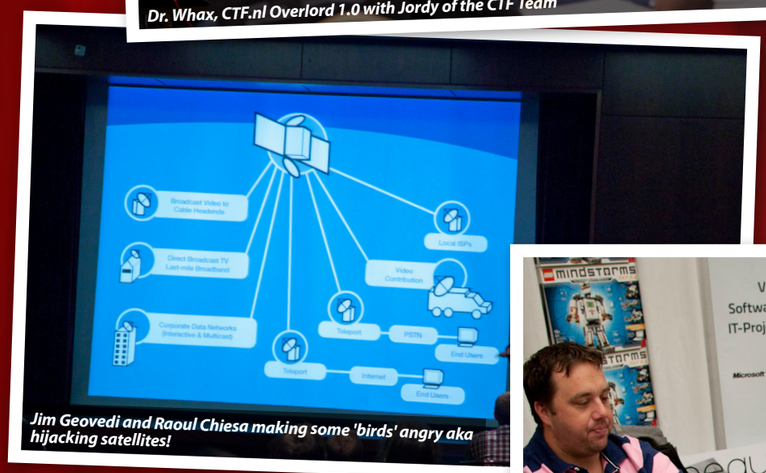
Dr. Whax, CTF.nl Overlord 1.0 with Jordy of the CTF Team



"Picking locks is good!"



Don't fear the Hax0r



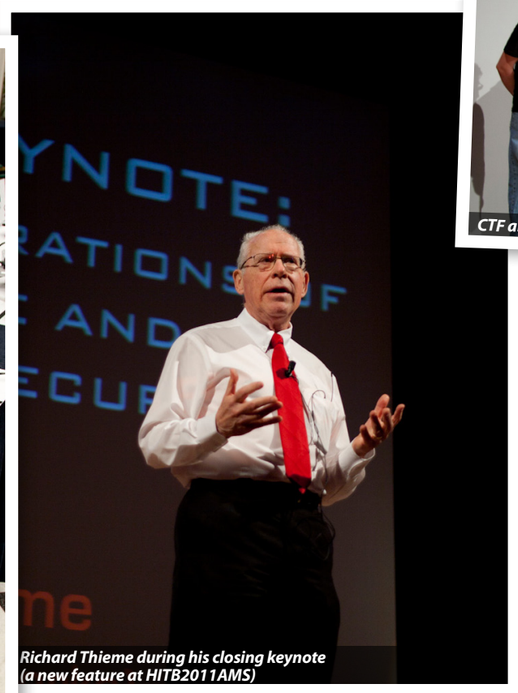
Jim Geovedi and Raoul Chiesa making some 'birds' angry aka hijacking satellites!



CTF and Hackerspaces Challenge winner announcement



Hackerspace Mindstorm bots ready for battle!


Richard Thieme during his closing keynote (a new feature at HITB2011AMS)


"A big warm THANK YOU to all our sponsors, speakers, crew, volunteers and of course attendees for joining us in making this A SUPERB conference indeed! See you at #HITB2011KUL in October!"

birds angry aka satellite hijacking! Day 2 also saw Adam 'Major Malfunction' Laurie and Daniele Bianco performing an EXCLUSIVE LIVE HACK of the Europay Mastercard Visa (EMV) credit card system, proving conclusively that it is well and truly broken! To wrap things up, Richard Thieme of THIEMEWORKS closed with an awesome thought provoking keynote!

Hearty congratulations to the CTF and ITQ LEGO Mindstorm challenge winners and of course HUGE THANKS to our sponsors, speakers, crew and volunteers for another fantastic event! See you in Malaysia for #HITB2011KUL this October! •

Event Website: *http://conference.hitb.nl/hitbsecconf2011ams/*
Event Materials: *http://conference.hitb.nl/hitbsecconf2011ams/materials/*
Event Photos: *http://photos.hitb.org/*

Random Data Gets In The Box

By **Nigel Brik** (***zkyp***)

Since HAR2009, a hacker festival/conference in The Netherlands, our little hackerspace in Utrecht, RandomData, has been quite close with the guys from Hack In The Box (HITB). I have to admit that I'd never heard of this security collective from Malaysia back then. We were talking about the conferences that they were giving in different places around the world and about them willing to come to The Netherlands for their next event. We were all excited.

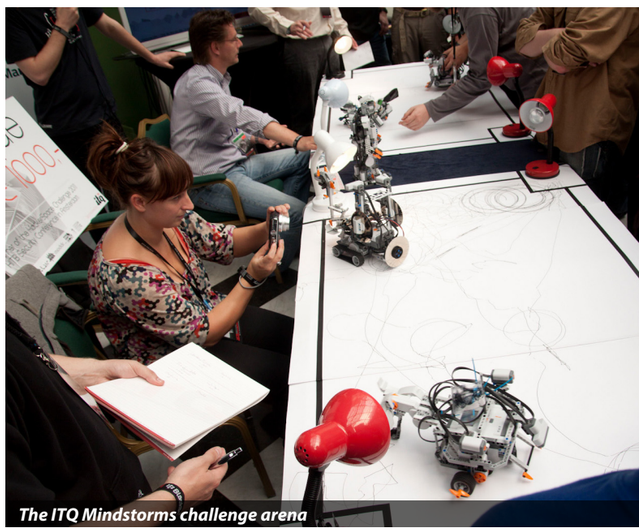
In 2010 the first HITBSecConf in Europe took place. Loads of guys from the hackerspace community, 2600NL and other friends of Randomdata + HITB joined up as volunteers to make this an experience to remember. For hackerspaces, there was a special area of the conference set-up to show off your projects which was visited by not only conference attendees but members of the public as well.

This year a lot of guys from the Dutch hackerspace community volunteered to make this another unforgettable experience. Because the guys behind HITB saw how enthusiastic the hackerspaces scene was to the event in 2010, this year they turned it up a notch. This year, in addition to the village there was also a hackerspace challenge sponsored by ITQ! No space knew what it was about or what to bring but after social engineering a bit, I found out that we were going to get to play with LEGO! Too bad my social engineering skills aren't that good, or I would've been able to found out more.

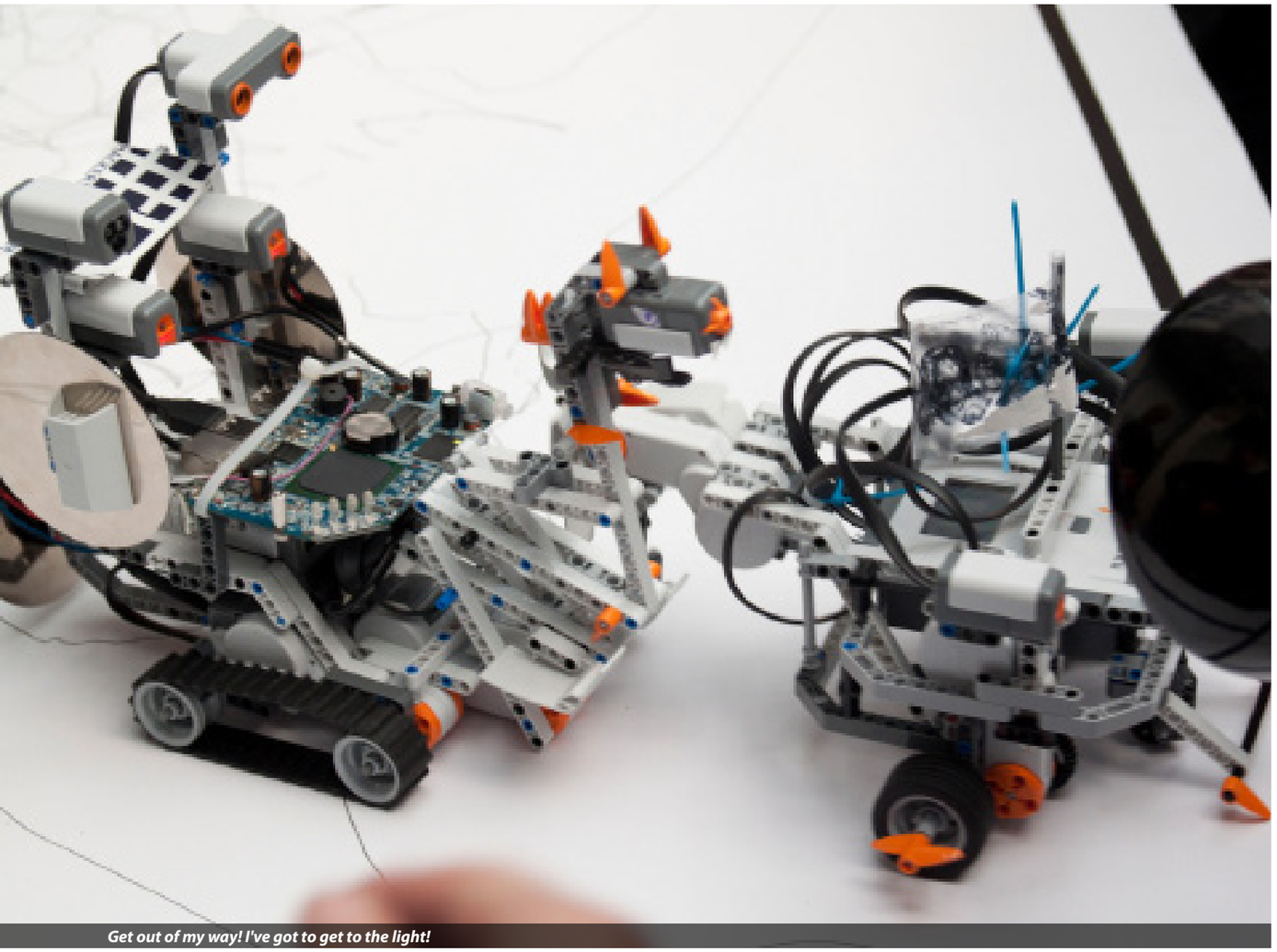
The challenge was awesome to say the least. We got to play with LEGO Mindstorm NXT's \o/! The challenge was to build a robot of some kind, using only the bits provided and the things that you brought with you to the event. Participants were not allowed to go out and buy stuff, only allowed to hack the stuff you had with you to build anything "extra". The ITQ stand had something which resembled a battleground - At a briefing of the



Our little LEGO bot



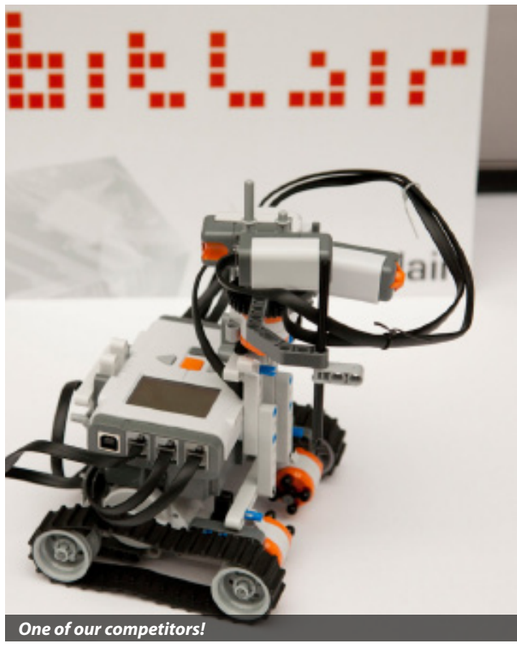
The ITQ Mindstorms challenge arena



Get out of my way! I've got to get to the light!



A birds eye view of a bot battle



One of our competitors!



We are the champions!

challenge, the objective was laid out - Teams would need to program their robot so that it would automatically drive to a light source which was placed on one of the four corners of the "battleground". The first robot to arrive would gain a point and this with a time limit of a few minutes. You could gain extra points by obstructing an opposing robot and also by having clean, robust code or a cool looking robot.

Because RandomData and HITB are close, most of our members are involved with the con in some way so it was a small problem to actually get guys to show off our (amazing and oh-so-many) projects! It was a good thing [com]buster was able to get time off work and was glad to join myself with the exhibiting. He also happens to be an excellent coder!

The building of the <robotname/pathfind>, was lots of fun and a good experience. It was cool to see what our hackerspace friends came up with and how they got there - Some started with the basics, others thought that the language provided by LEGO was inferior and started by making the NXT brick speak a different language. I saw another hackerspace who just started to build a dragon out of it. Our road was less spectacular. We just wanted to get the robot working with all the different sensors so it would be able to compete in the challenge, then worry about arming ourselves for the obstruction bonus points. We also only had five hours on day 1 and three the next to get this done!

By the afternoon of day 2, every participating space had a working robot and proudly set out to compete in the challenge! At this point, we found that our robot was actually doing very well. We saw that some robots were using sensors for the black lines at the end of the field, so they would know where to stop. Fifteen minutes before the start of the challenge we thought up a little idea; To add black markers to the side of our robot which would write on the ground, whereever we went! The idea was good but the lines were too thin - the lines our robot made could perhaps instead be sold as art!Another idea we had was to build a light dome on top of our robot. Seeing that the objective was too be the first at the light, we thought this might sidetrack some robots. After some soldering and failing, we saw that Bitlair was building a bulldozer-like robot which would 'pick up' anything in it's path - We decided we should add some extra lego-bar protection instead of a lightdome.

After thirty minutes of battling, the challenge was done and after some quick math by the ITQ judges, RandomData was pronounced the winner! Huzzah! 1000 EUR for the win! Bitlair and their bulldozer bot came second and whitespace(0x20) from Gent, Belgium came third.

Overall, it was a a great event and we're already looking forward to HITB2012AMS! •



Next Generation Web Attacks – HTML 5, DOM (L3) and XHR (L2)

Shreeraj Shah, *Blueinfy Solutions*

Browsers are enhancing their feature-sets to accommo- date new specifications like HTML 5, XHR Level 2 and DOM Level 3. These are beginning to form the backbone of any next generation application, be it running on mobile devices, PDA devices or desktops.

The blend of DOM L3 (Remote Execution stack), XHR L2 (Sockets for injections) and HTML5 (Exploit delivery platform) is all set to become the easy stage for all attackers and worms. We have already witnessed these types of attacks on popular sites like twitter, facebook or yahoo. Hence the need of the hour is to understand this attack surface and the attack vectors in order to protect next generation applications. Moreover this attack surface is expanding rapidly with the inclusion of features like audio/video tags, drag/drop APIs, CSS-Opacity, localstorage, web workers, DOM selectors, mouse gesturing, native JSON, cross site access controls, offline browsing etc. This expansion of attack surface and exposure of server side APIs allows the attacker to perform lethal attacks and abuses such as:

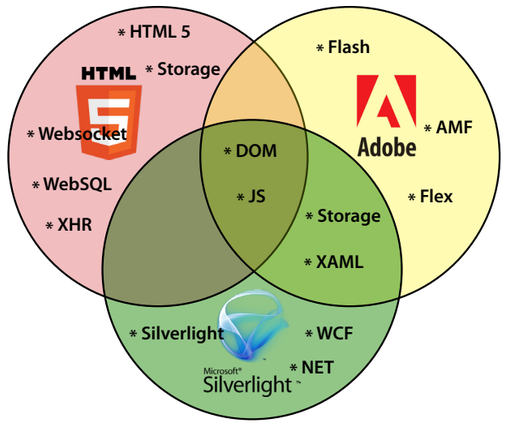
- XHR abuse alongwith attacking Cross Site access controls using level 2 calls
- JSON manipulations and poisoning
- DOM API injections and script executions
- Abusing HTML5 tag structure and attributes
- Localstorage manipulations and foreign site access
- Attacking client side sandbox architectures
- DOM scrubbing and logical abuse
- Browser hijacking and exploitations through advanced DOM features
- One-way CSRF and abusing vulnerable sites
- DOM event injections and event controlling (Clickjacking)
- Hacking widgets, mashups and social networking sites
- Abusing client side Web 2.0 and RIA libraries

HTML 5 ON THE RISE – RESHAPING THE RIA SPACE

Web applications have traveled a significant distance in the last decade. Looking back, it all started with CGI scripts and now we are witnessing the era of RIA and Cloud applications. Also, over these years existing specifications evolved to support the requirements and technologies. To cite an instance, in the last few years Flex and Silverlight technology stacks have not only come up but also continued to evolve to empower the browser to provide a rich Internet experience. To compete with this stack the browser needed to add more native support to its inherent capabilities. HTML 5, DOM (Level 3) and XHR (Level 2) are new specifications being implemented in the browser, to make applications more effective, efficient and flexible. Hence, now we have three important technology stacks in the browser and each one of them has its own security weaknesses and strengths (*Figure 1*).

HTML 5 has caused the underlying browser stack (application layer especially) to change on many fronts. Moreover, it has added the following significant new

Figure 1. Technologies running inside the browser stack



components to support application development.

- Support for various other technology stacks through plugins (Silverlight and Flash)
- New tags and modified attributes to support media, forms, iframes etc.
- Advance networking calls and capabilities from XMLHttpRequest (XHR) object – level 2 and WebSockets (TCP streaming).
- Browsers' own storage capabilities (Session, Local and Global)
- Applications can now run in an offline mode too by leveraging the local database which resides and runs in the browser, known as WebSQL.
- Powerful Document Object Model (DOM – Level 3) to support and glue various browser components and technologies.
- Sandboxing and iframe isolations by logical compartments inside the browser.
- Native support in the browser or through plugins for various different data streams like JSON, AMF, WCF, XML etc.
- Drag and Drop directly in the browser made possible to make the experience more desktop friendly.
- Browsers' capabilities of performing input validations to protect their end clients.

HTML 5 – EXPANSION OF ATTACK SURFACE AND POSSIBLE ABUSES

HTML 5 with its implementation across the browsers has given a new face to the threat model. There are various new openings and entry points that lure an attacker to craft variants for existing attack vectors and successfully abuse the security. As show in *Figure 3* the several components of

Figure 2. Browser stack with HTML 5

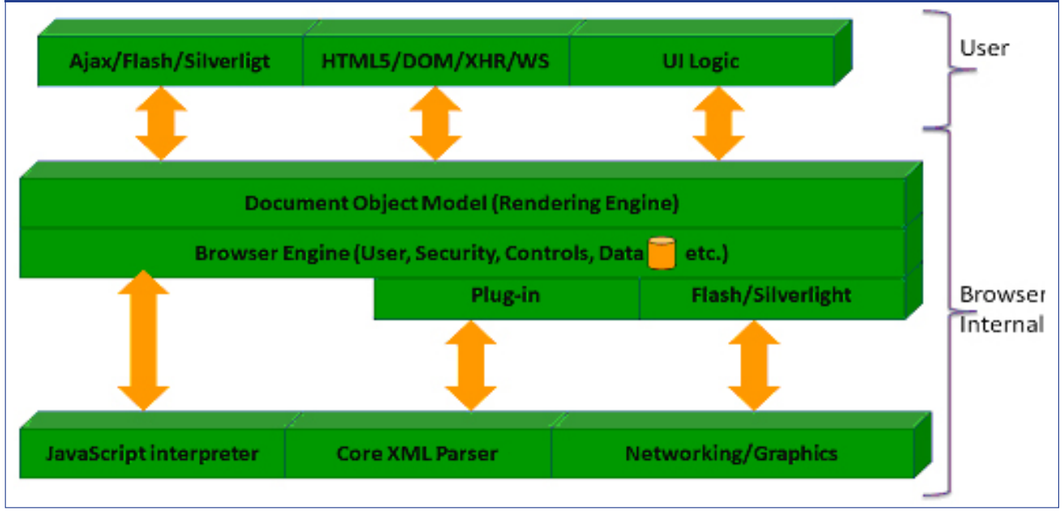
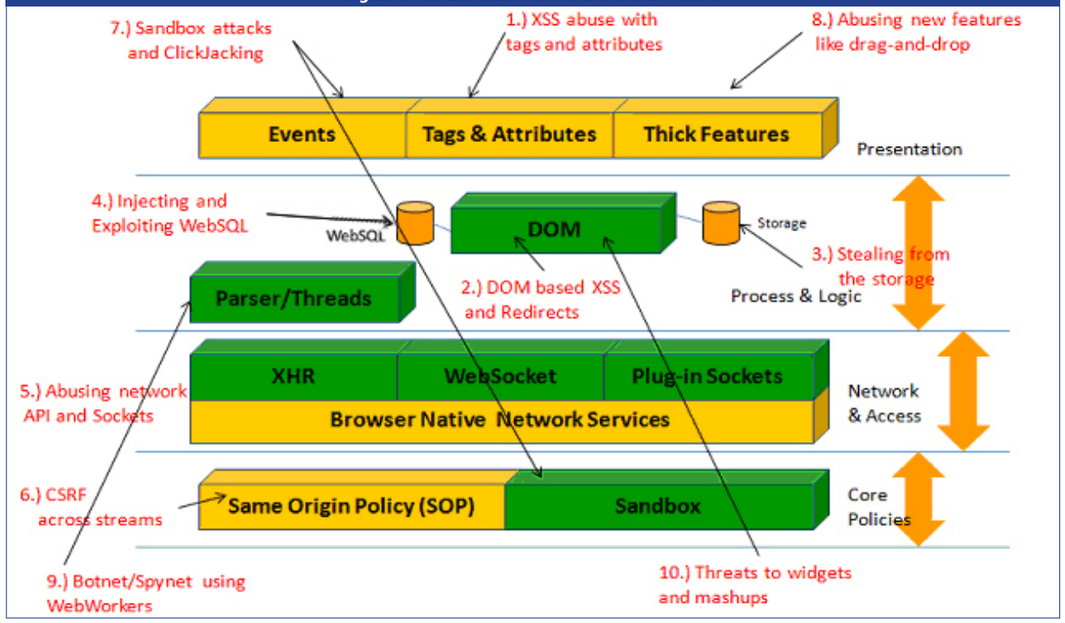


Figure 3. HTML 5 attack surface and attack vectors



HTML 5 can be divided into four segments – presentation, process/logic, network access and policies.

- Enhanced event model, tags, attributes and a thick set of advanced features can cause the crafting of attack vectors like ClickJacking and XSS
- DOM and browser threads can be abused with DOM based XSS, redirects, widgets/mashup attacks
- Storage and WebSQL can be exploited by poisoning and stealing the same
- WebSockets, XHR and other sockets can be abused too
- Same Origin Policy (SOP) can be attacked with CSRF using various streams

Based on the above threat model and attack surface synopsis the following are some interesting attack vectors.

AV 1 - XSS ABUSE WITH TAGS AND ATTRIBUTES

HTML 5 has added extra tags and attributes to support various new features and functionalities. For example one can add simple **'media' tags** to add video and audio across web pages. HTML **forms** have also been updated and provide new attributes. All these new tags and attributes allow triggering of JavaScript code execution.

As a result, if parameters going to these tags and attributes are not duely validated then XSS is a natural easy fallout – persistent as well as reflected. These new components of HTML 5 help in bypassing existing XSS filters which have not been updated to keep their eyes on these newly added tags. Hence, by carefully analyzing the new tags and their behavior, an attacker can leverage these newly added mechanisms and craft possible exploits to abuse HTML 5.

Consider the following examples:

Abusing media tags: The following are some interesting injections possible in media tags. A set of browsers have been seen to be vulnerable to this category of attack variants. Both audio and video tags are vulnerable to possible abuse.

```
<video poster=javascript:alert(document.
cookie)//

<audio><source onerror="javascript:alert(doc
ument.cookie)">
```

Injection within form attributes like 'formaction', 'autofocus' or 'oninput': This can also result into XSS:

```
<form><button formaction="javascript:alert(d
ocument.cookie)">foo

<body oninput=alert(document.cookie)><input
autofocus>
```

On a similar basis, there are a few other tags that can also be abused and attacked.

AV 2 - DOM BASED XSS AND REDIRECTS

Document Object Model (DOM) is an integral part of the web browsers using which the content is rendered. Web applications use DOM to **manage the presentation layer** of the application. It allows the browser side application to make Ajax calls using XHR and render new content as and when required within existing placeholders say "div" positions. All new libraries and JavaScripts use DOM extensively as they make DOM calls for a variety of functionalities.

DOM has been enhanced to support HTML 5 and XHR with the implementation and inclusion of new features which are beginning to be used by the next generation apps extensively (*http://www.w3.org/TR/DOM-Level-3-Core/changes.html*). DOM supports features like XPATH processing, DOMUserData, Configuration etc. Web applications use the DOM for stream processing and various different calls like document.*, eval etc. If an application uses these calls loosely then it can fall easy prey to potential abuse in the form of XSS. Also, the browser processes parameters from the URL separated by hash (#), allows values to be passed directly to the DOM without any intermediate HTTP request back to server, allows off-line browsing across local pages and database and allows injection of potential un-validated redirect and forwards as well. In view of all this, DOM based XSS are popular vulnerabilities to look out for, when it comes to HTML 5 driven applications.

Consider the following examples:

Document.write causes XSS:

```
if (http.readyState == 4) {
        var response = http.responseText;
        var p = eval("(" + response + ")");
            document.open();
            document.write(p.
firstName+"<br>");
            document.write(p.lastName+"<br>");
            document.write(p.phoneNumbers[0]);
            document.close();
```

Here is a list of few other calls which can cause XSS if the parameter stream comes from any untrusted source.

```
document.write(…)
document.writeln(…)
document.body.innerHtml=…
document.forms[0].action=…
document.attachEvent(…)
document.create…(…)
document.execCommand(…)
document.body. …
window.attachEvent(…)
document.location=…
document.location.hostname=…
document.location.replace(…)
document.location.assign(…)
document.URL=…
window.navigate(…)
document.open(…)
window.open(…)
window.location.href=…
eval(…)
window.execScript(…)
window.setInterval(…)
window.setTimeout(…)
```

Redirect through DOM itself (via location):

For example, in a case as follows
http://foobank.com/app/#http://www.evilsite.com/

* gets processed within the DOM and the resultant '*http://www.evilsite.com/*' if passed to a location call at some point would result into a successful attack.

AV 3 - STEALING FROM THE STORAGE

W3C has come up with a new specification for web clients in HTML 5. This is to lay the ground work to have local storage for a website (*http://www.w3.org/TR/webstorage/*). Interestingly, according to this websites are allowed to create a nice array for variable storage in their own sandbox. This is bound by document.domain context. Hence, it is not possible to bypass a sandbox and access a foreign site's storage information (say a cookie). Here is the interface for the storage:

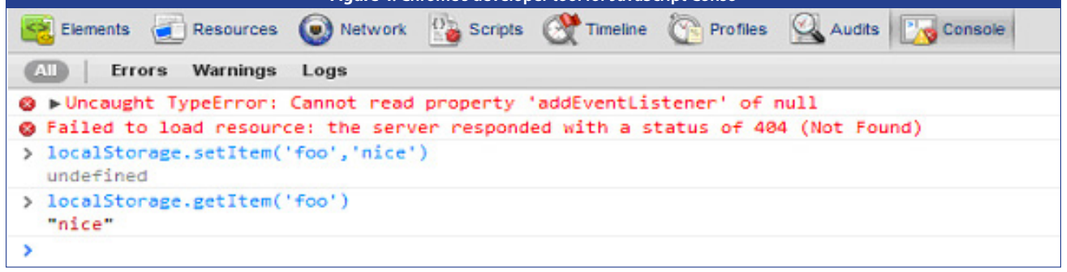
```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long
index);
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString
key, in any data);
  deleter void removeItem(in DOMString key);
  void clear();
};
```

Citing an example,

Any domain can set values using JavaScript. Here is a simple example of setting and retrieving values from local storage.

```
openDatabase
executeSql
```

Figure 4. Chrome's developer tool for JavaScript Conso



In this scenario, the following are the key threats to the Local Storage Mechanism, which one needs to address before implementing this functionality in an application.

- **DNS spoofing** - By way of DNS spoofing an attacker can gain access to the stored information from the browser. If any sensitive information has been stored you run the risk of identity and privacy. This can be avoided by serving over an SSL channel so that the DNS is locked to the certificate and not an easy prey.

- **XSS attack** – XSS can scrub the local storage and access juicy information if available. It is important to note that although the HTTPOnly cookie cannot be accessed by the script the session id stored on Local Storage can be accessed via XSS.

- If the application is running a hierarchical domain structure and these domains are owned by different authors, there is potential for compromise. It may be possible to access local storage information on the basis of the parent domain which might be common amongst various child domains.

AV 4 - INJECTING AND EXPLOITING WEBSQL

HTML 5 also provides support for light database functinality within the browser. This allows applications to use and dump information on the local machine. This in turn makes the application effective and fast in some cases. At the starting point, the application can write to this database following which it is allowed to make local calls to the database from the browser itself. Its speed is enhanced here since the application can fetch data without the need of an HTTP call and response two way interaction with the server.

The following are the calls to access the database:

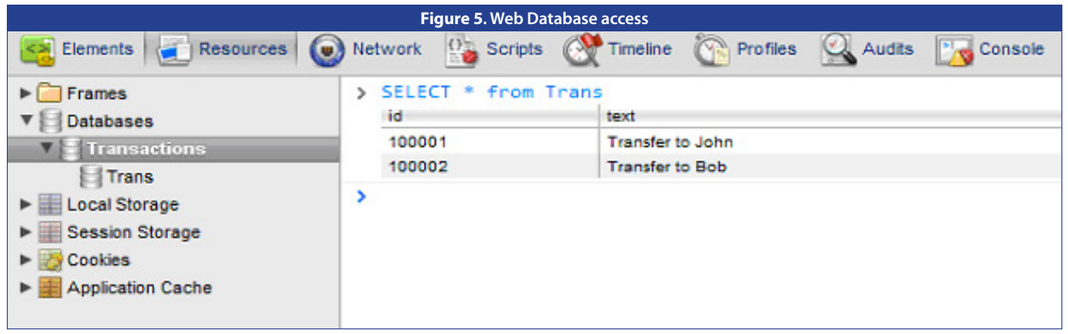
These facilitate database creation as well as query execution. Here is a view of chrome where you can see the db and run queries as well.

If an application is using this HTML 5 feature then the potential threats to be kept in mind are as under:

- If a part of the application is compromised by XSS then the attacker can get both accesses to this database – read and write. Hence, it is possible to change as well as read values from the target table.

- It is also possible to perform client side SQL injections and bypass some business logic as well.

Figure 5. Web Database access



- Also if the database is being used in the offline fashion; it can be compromised by an attacker who can fetch access.

Hence, there are several potential threats to the use of this web database enhancement. As a result one needs to tread with care with the type of data being handled using these calls.

AV 5 - ABUSING NETWORK API AND SOCKETS

Sockets are always crucial since they are great targets from the attacker's perspective. Malware, Spyware and XSS vectors use sockets frequently for several purposes. HTML 5 supports WebSockets and advanced XMLHttpRequest (XHR) Level 2 calls. This offers a variety of options to the attacker as malicious code can be channeled back to target systems. WebSockets can be effectively used for TCP scanning and communications also.

Consider the following example:

Here we are trying to scan port 80 and catch the response back using WebSockets.

WebSocket has its own event model. Ready state can be used to determine TCP ports and for other analysis as well. This allows calls to be made across domains as well. Both XHR and WebSocket open up security concerns. Here are a few of these concerns and possible attacks:

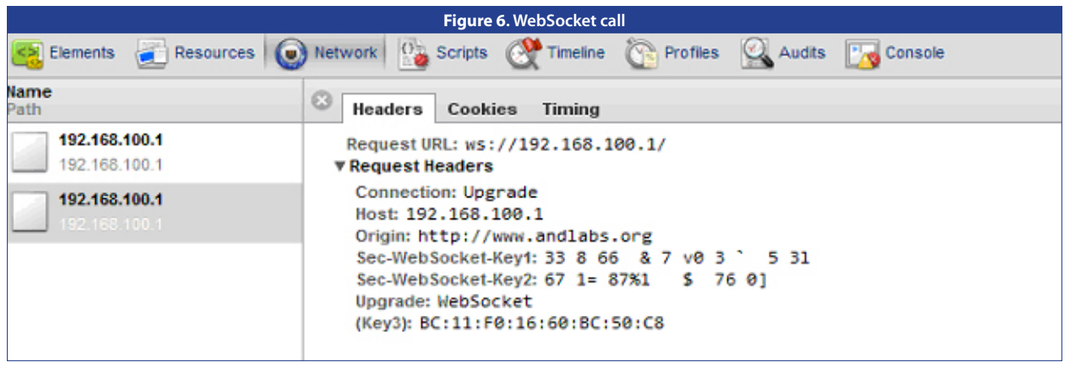
- An attacker can force internal port scanning, IP detection and full blown exploitation across the network through the browser. It can be lethal since the attacker who could not go through the firewall can now use this backdoor to enter internal networks.
- The attacker can also use these sockets to establish a backward channel to his own system once the browser has been compromised.
- Sockets talk to proxy and cache, which opens up another set of security concerns and allows an attacker to divert traffic.

Hence, with HTML 5 we have to address these new threats and consider them a part of our threat model. These can pose a serious threat to the application layer running inside the web browser.

AV 6 - CSRF ACROSS STREAMS – JSON, AMF AND XML

Cross domain calls are a major concern from the security perspective. The browser has its own SOP (Same Origin Policy) in place to avoid cross domain calls. Many times these calls replay the cookie and make the HTTP calls context sensitive binding identity along with the calls.

Figure 6. WebSocket call



There are several tags like script or iframe which originate these cross domain calls but through Ajax they are a bit restricted. Browsers have implemented mechanisms for these. HTML 5 has come up with the postMessage() mechanism which allows frames to communicate with cross or same domains provided that the events are registered.

```
interface MessageEvent : Event {
  readonly attribute any data;
  readonly attribute DOMString origin;
  readonly attribute DOMString lastEventId;
  readonly attribute WindowProxy source;
  readonly attribute MessagePortArray ports;
  void initMessageEvent(in DOMString
typeArg, in boolean canBubbleArg, in boolean
cancelableArg, in any dataArg, in DOMString
originArg, in DOMString lastEventIdArg, in
WindowProxy sourceArg, in MessagePortArray
portsArg);
};
```

If an application does not check the actual "origin" of the call then it can be seen as a potential security issue.

CSRF can be caused via various streams and not restricted to typical name/value pairs on GET/POST requests. HTML 5 and RIAs use various structures like JSON, XML , AMF etc. All these can be polluted with CSRF. One can force the browser to originate these streams and attack CSRF entry points. Security concerns are also observed on the proxy running on the server side to allow cross domain content sharing.

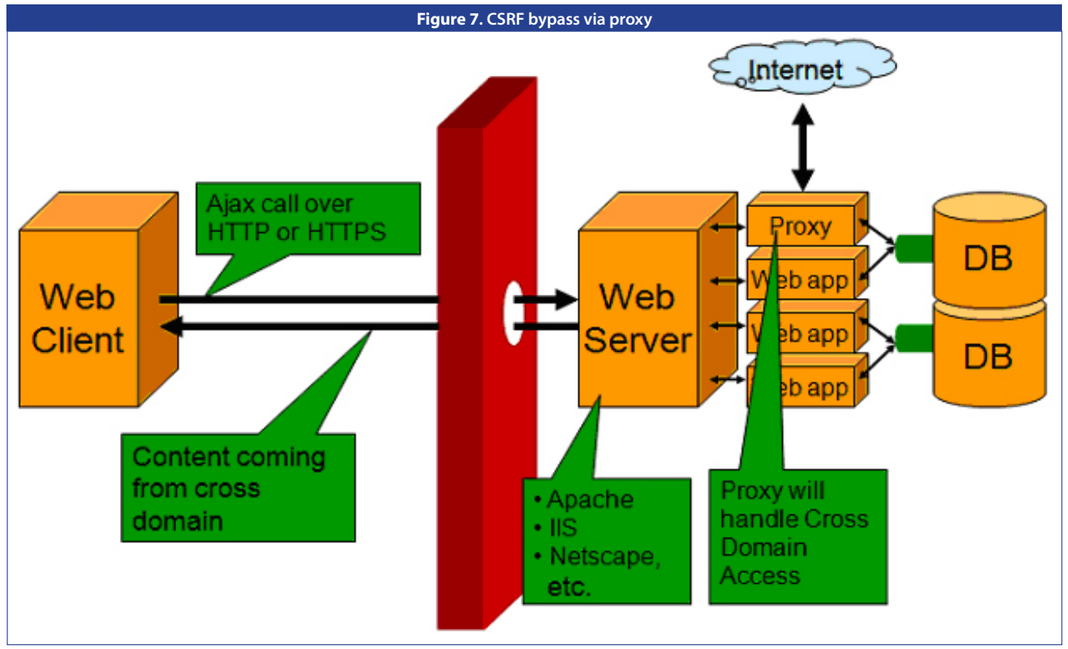
For example,

AMF stream Injection:

```
<html>
<body>
<FORM NAME="buy" ENCTYPE="text/plain" act
ion="http://192.168.100.101:8080/samples/
messagebroker/http" METHOD="POST">
      <input type="hidden" name='<amfx ver'
value='"3" xmlns="http://www.macromedia.
com/2005/amfx"><body><object type="flex.
messaging.messages.CommandMessage"><trait
s><string>body</string><string>clientId</
string><string>correlationId</
string><string>destination</
string><string>headers</
string><string>messageId</
string><string>operation</
string><string>timestamp</
string><string>timeToLive</string></
traits><object><traits/></object><null/><str
ing/><string/><object><traits><string>DSId</
string><string>DSMessagingVersion</string></
traits><string>nil</string><int>1</int></
object><string>68AFD7CE-BFE2-4881-E6FD-
694A0148122B</string><int>5</int><int>0</
int><int>0</int></object></body></amfx>'>
</FORM>
<script>document.buy.submit();</script>
</body>
</html>
```

CSRF and cross domain bypass could be considered one of the major security threat aspects of HTML 5 and in the future we may see some innovative bypasses to abuse

Figure 7. CSRF bypass via proxy



these new functionalities.

Also, if the browser supports auto setter for JSON, this can lead to two way CSRF where content can be read as well. Some of the browsers and mobile devices allow JSON literals which can be controlled by user input, which in turn triggers at the point of stream processing and it's possible to overload. This allows a user to get access to a JSON object or array. This is another possible vector to manipulate JSON based processing if implemented in an incorrect fashion.

AV 7 - SANDBOX ATTACKS AND CLICKJACKING

ClickJacking or UI regressing is an interesting vector emerging on the net. After the introduction of social networking sites, it seems to have become a popular attack vector to cause malicious events from legitimate sessions. HTML 5 allows various ways of enhancing the GUI inside a browser. HTML 5 has brought along the introduction of new tags like canvas. CSS enhancement ability allows ClickJacking attack vectors to be formed relatively easily. Also browsers have introduced the sandboxing ability which allows reverse ClickJacking where an attacker can load his domain on the frame, being on the same domain, by leveraging vulnerabilities like XSS. It allows the attacker to stay persistently on the site and monitor all moves made by the end user as well as retrieve information from his session.

Iframe has been another potential place for abuse within the browser stack all these years. It is a feature to host cross domain content within the current page. It allows cross domain calls and can be abused by forcing Clickjacking.

New specifications have come up with a mechanism to provide a sandbox across the browser's iframe. Some of these browsers have implemented these as well but those instances can also be abused in a scenario as under:

```
<iframe sandbox="allow-same-origin allow-
forms allow-scripts" src="http://www.foobar.
com/"></iframe>
```

If the application is using JavaScript driven frame-bursting solution to protect against ClickJacking then the above tag can help in abusing the functionality in some cases – say for example "allow-top-navigation" parameter.

AV 8 - ABUSING NEW FEATURES LIKE DRAG-AND-DROP

HTML 5 has some interesting innovative methods, events and tags to make the browser application very rich in look and feel. This includes functionalities like drag and drop so one can communicate from the desktop using just the mouse. The browser captures these events and fires backend calls. Unfortunately, this mechanism can be abused easily. It is possible to exploit this, by transferring malicious code by injections into **setData** via **draggable (true)** and firing event at **ondragstart**.

For example,

```
<div draggable="true" ondragstart="event.
dataTransfer.setData('text/plain','code
injection');">
```

It is possible to transfer malicious code at the point of the event.

AV 9 - BOTNET/SPYNET GETS PERSISTENT LIFE USING WEBWORKERS

WebWorkers are a new introduction in the specification. This functionality allows the browser to run scripts in the background along with the main page. This effectively makes the browser similar to a multithreaded application and one can leverage this method.

For example here is a simple worker.js script which we are running in the background. It can use postMessage to report back as well.

```
<script>
   var w = new Worker('worker.js');
   w.onmessage = function (event) {
     document.getElementById('myresults').
textContent = event.data;
   };
</script>
```

This can be leveraged by spinet and botnet as well. They usually load their script through the iFrame or script tag but here is a different way to load the code. They can load it using webworker and stay on the page in a hidden fashion. This makes their detection difficult for monitoring tools as well.

AV 10 - THREATS TO WIDGETS AND MASHUPS

In many applications one can inject a Widget or a Gadget. This little HTML code along with JavaScript runs on the DOM and performs several operations. In some cases these Widgets share the same DOM or a part of the DOM. This may also allow one Widget to access the important tags and variables of another Widget. In these cases, an attacker can force a malicious widget on the DOM and monitor other widgets.

For example, consider a Widget which takes the username and password credentials. Here is a simple way in which another Widget can set a trap on it.

Figure 8. Setting a trap

```
4  function regEvent4me()
5  {
6      var objs = document.getElementsByName("txtUser");
7      if (objs.length > 0)
8      {
9          var thefield = objs[0];
10         thefield.onblur = GetU;
11     }
12     objs = document.getElementsByName("txtPass");
13     if (objs.length > 0)
14     {
15         var thefield = objs[0];
16         thefield.onblur = GetP;
17     }
18 }
```

Hence, a malicious widget is listening to the mouse event and as soon as the credentials are entered it can force GetU and GetP function calls to be made. These functions can go ahead and steal the content and send it across the network to a place where the attacker may be listening. Evidently, It is important to analyze the DOM architecture and usage when it comes to Widget platforms.

CONCLUSION

The scenario of the web and the upcoming technology stacks bears strong resemblance to the thief police scenario. As the web world progresses, the demands of users are matched by stronger and richer functionalities growing by the day. The flipside of this coin is that as such enhancements come about, loopholes and vulnerabilities increase with the extended attack surface. HTML 5, DOM L3 and XHR L2 are a combination of this same kind. With the enormous enhancement in the look and feel of the web applications that they have brought, the attackers are also not likely to remain inactive. Thus along with bearing the advantages of these technologies it is of prime importance that we tread carefully and become aware to the new threat model and work on countermeasures. This can be the only way to bear the advantages and yet not be bogged down by the attacks i.e. without loss of privacy and security. •



Botnet-Resistant Coding

Web malware infections are proliferating, and the online banking industry has become the hottest target. Stealthy bots play a critical role in the success of these attacks. In this paper, we propose a new approach to mitigating the impact of botnet infections.

Fabian Rothschild and Peter Greko, *Hack Miami*
Aditya K Sood and Richard J Enbody, *Michigan State University*

INTRODUCTION

Bots are compromised (or victim) computers and a botnet is an organized collection of bots. A botmaster controls a botnet through a command-and-control (C&C) center. A typical scenario is the use of a Trojan program to infect (compromise) a computer with malware that will communicate with the C&C center. Botnets can be used to collect data from compromised machines or to use the bots collectively for tasks such as spamming or denial-of-service attacks.

Botnets[1,2] have been infecting the Web for a few years, but recently there has been a dramatic increase in both the size of the botnets and the malicious operations performed by them. Of particular interest are the fraud and money laundering activities because of the financial damage they can cause. Over time botnets have become more sophisticated; the Zeus bot[3] is a recent example. Unfortunately, no prevention mechanism exists that can be used in line with existing applications in order to prevent stealing of data by them.

In this paper, we focus on creating botnet-resistant code that works under the assumption that client machines are already infected. Our concept is a result of a number of experiments we have performed to directly build in defenses within client applications. Our approach is new in that we exploit techniques used by the botmaster to harvest information. By understanding and corrupting the botmaster's processes we can disrupt their information-stealing techniques. For our experiments we targeted the Zeus botnet.

ART OF HARVESTING INFORMATION – BOTMASTER PSYCHOLOGY

Botmasters are effective at subverting the running environment of victim machines to perform unauthorized operations such as stealing banking information. Botnets follow a typical lifecycle to steal information from infected machines. For example, there exists a Botnet Business Cycle (BBC) in which stolen data is sold across different domains through an intermediate party called as guarantor. The sensitive information includes credit card numbers, logins, social network credentials and email logins that are needed by criminal customers of the guarantor in order to initiate targeted attacks. *Figure 1* shows a high-level view of a BBC model, and *Figure 2* shows the alerts that are used by a botmaster to advertise on underground forums.

Botnets also collect a wide variety of other information from infected machines including usernames, passwords, cookies, view state parameters: everything which is passed as form values using POST requests during submission of forms. *Figure 3* shows the lifecycle of a botnet log storing process: (1) infect (2) log victim data on the server (3) harvest log.

Infected machines communicate with a Command and Control (C&C) server, sending victim information back and receiving instructions. In its simplest form the C&C is nothing but a PHP based application that serves as a framework for managing botnet activities. The C&C may support a backend database to make the analysis of collected data easier. Data on victim machines are usually aggregated into logs in plain text so the botmaster can harvest information using pattern matching and simple data mining techniques. Zeus collects its logs on the server side.

Figure 1. Money Flow Hierarchy – Botnet Business Cycle

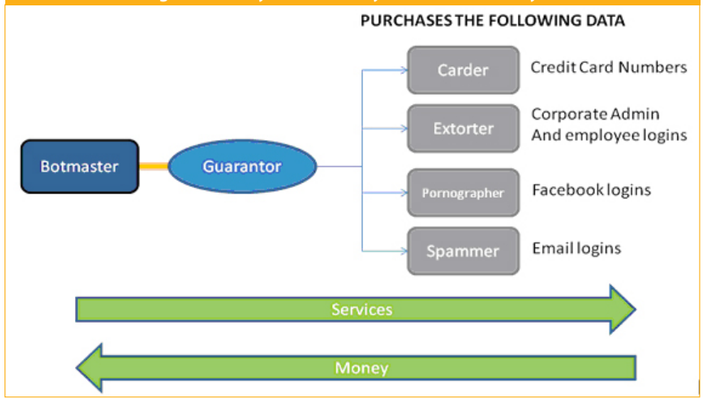
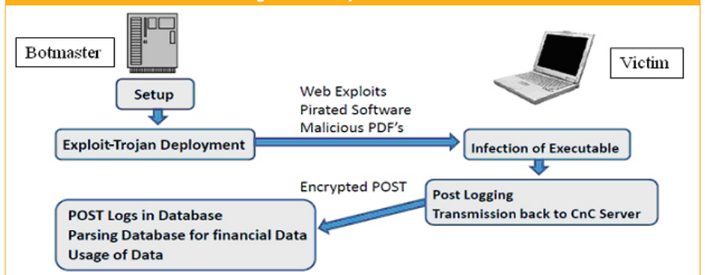


Figure 2. Notifications and Alerts on Forums



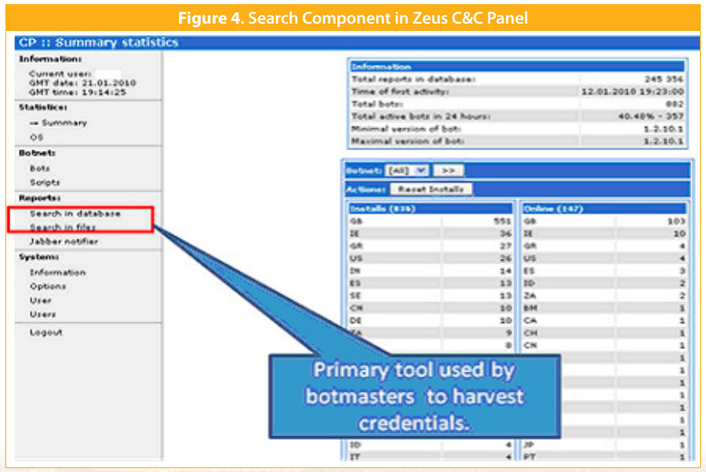
Figure 3. Lifecycle of a Botnet



APPROACH – BOTNET RESISTANT CODE

The goal of botnet resistant coding is to disrupt the botmaster's activities using its own tools and techniques. The baseline of our approach is to make the log harvesting process harder for the botmaster. During our experiments on Zeus, we observed that the database that resides in the C&C server encounters a high volume of traffic in the form of logs carrying sensitive information. Considering the purchase and sale of data in the underground economy, if the sensitive information is hard to find in the harvested data, it can reduce the incentives to engage in data selling and stealing activities. Making the process harder to harvest the data and downgrading the quality of the data also affects the financial drive to spend the time to create and build a botnet. As the log data is present in a raw form, it is not easy to find the credentials (username, password, credit card numbers, etc.). The botmaster has to apply data mining techniques to extract the sensitive information from the logs. For example: the generic case is to look for combinations of "username" and "password" variable names in the forms and their respective values. Another way is to look for a generic variable name for credit cards such as "cc_number" that are used in a number of bank applications. *Figure 4* shows the Zeus C&C panel displaying search functionality to find credentials in the logs.

Figure 4. Search Component in Zeus C&C Panel



Before continuing with a discussion of potentially disruptive techniques, it is worth clarifying the impact of HTTPS that is used to secure Web communication. HTTPS cannot protect victims from malware installed on their machines such as Zeus infections. The reason is that Zeus undermines the encryption process by stealing the data before the data is actually sent over the wire. Examples of this type of theft include keylogging and in-memory modification of the functions that carry out the encryption. Botnets such as Zeus capture only the victim's POST requests data and do not care about the GET requests. As we show later, we have exploited this functionality of the Zeus botnet in order to pass encryption keys and mangling functions through GET requests. In this way we can add functionality to the browser for our disruptive activities in a way that will be "under the radar" of Zeus.

In order to differentiate among different approaches of coding that are designed for mitigating botnet infections, we have divided the process into different levels. In every level, we are going to talk about the impact of the keylogger and the respective log storage. The main goal is to disrupt the data logging and harvesting processes.

- **Basic:** In this form of prevention, we are primarily interested in manipulating the name of variables that are used for credentials and other sensitive information. HTTP servers do not care about the name of variables as long as the protocol works appropriately. This form of prevention is for servers which don't have the bandwidth or processing power to deal with more intensive prevention methods. This is a minimal attempt to make data harder to harvest and thereby making the data less valuable.

- **Medium:** In this form of prevention, JavaScript functions modify variable names by introducing post fixing, prefixing and data mangling

functions. The resulting obfuscation will be easy for a bank server to undo, but very difficult for the botnet. As a result, it becomes hard for botmaster to carry out the analysis on mangled data. In addition, fake data can be added to bloat the logs to create more work for the botmaster and to further obfuscate the data.

• **Hard:** This level introduces server side sessions and JavaScript with AJAX methods. Some experiments have shown that bots do not log data for typical AJAX requests. Further, it is also possible to introduce fake data functions and symmetric encryption to tackle POST requests differently

It is important to consider bank server load when weighing the benefits of botnet resistant coding. Server costs can increase tremendously when encryption and excessive POST requests are sent to the servers. Application load was tested on several server and desktop machines running LAMP stacks, and load increased with the degree of botnet-resistance of coding. However, client machines showed little to no overhead upon the regular web browser performance hit.

Basic Level

The goal of the basic prevention coding practices is to make the botmaster's job a little harder. This solution targets the logs associated with the botnet and the credentials that are being harvested. Methods listed in this section are focused on preventing the attacker from searching for victim data that are easily recognizable. Most botnet logs carry large amounts of harvested data captured from the infected victim machines. Sifting through the collected data can be a challenge at times: the faster the botmaster harvests the credentials, the faster they can turn them into usable cash. Most search queries on the C&C panel involve keywords such as "Username" and "Password". Botnet logs are often sold in Mega Bytes (MB) and the price reflects the amount of data along with the quality of data.

Figure 5. Basic Prevention technique in practice

Bot ID:	**sheep_a_deep_0056ee58**
Botnet:	-- default --
Version:	1.3.1.1
OS Version:	XP Professional SP 3, build 2600
OS Language:	1033
Local time:	22.03.2011 23:18:33
GMT:	-4:00
Session time:	29:27:18
Report time:	23.03.2011 03:18:42
Country:	--
IPv4:	127.0.0.1
Comments for bot:	-
In the list of used:	No
Process name:	C:\Program Files\Internet Explorer\iexplore.exe
User of process:	-
Source:	http://172.17.17.115/simple/success.php

```
http://172.17.17.115/simple/success.php
Referer: http://172.17.17.115/simple/
Keys: 2324524545435
Data:

real data=2394857239485729345702394857290548 2-398542-9854-750293847502938475
not the real account data dont look=2324524545435
srsly real data=2394857239485729345702394857290548 2-398542-9854-975029384750293847 5
```

The basic method obfuscates variable names. Variable names in form fields do not have to be "UserName" or "Password." They can be any string as long as it is understood on the bank server side. Typical botnet users, as demonstrated earlier, look through logs using string queries for such phrases as "CVV", "UserName", "Password", or "Address". Obfuscating variable names can be accomplished in the design of the bank websites. That is, different banks can use their own variable names that are based on their own policies so that log monitoring remains easy for them, but simple search is made more difficult for the botmaster. This approach can be a very simple fix and can be implemented on most web-based applications with few changes done to the code base.

In *Listing 1*, the username input field is called "ALH84001" and the password field is called "NASA_AMS". This approach will bypass simple queries for username and password fields in the C&C. For a botmaster to counteract this approach, a custom query would have to be made for these types of variables to find the login credentials generated from this site. Another method involves hidden form fields. These hidden form fields are transparent to the user and can be used to send false information to make data harvesting more difficult. *Figure 5* shows an output of the experiment conducted using the Zeus bot.

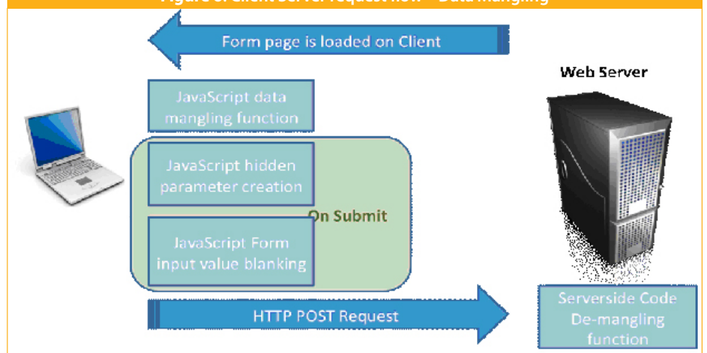
This method is simple to implement without requiring any rewriting to the base server side code. Simple HTML form changes can easily be done without harming any of the server side code. The server deals with the load of the hidden parameters which is almost inconsequential.

Medium Level

This prevention method uses JavaScript based modules to implement obfuscation in POST requests. As mentioned above, the Zeus botnet does not record GET requests. This fact allows us to pass obfuscation functions via JavaScript files to the client without this activity being recorded in the botnet logs. When an infected user clicks on the submit button, all the form parameters are obfuscated using the GET-supplied obfuscation function and then sent as a hidden parameter. With this technique, all other form field values are blanked out or filled with false data and sent in the normal POST request. Unfortunately, Zeus will still log all this data since both the obfuscated and false data is in the POST. However, the obfuscation increases the difficulty of Zeus harvesting its logs. Of course, the bank's server side code will have to unwrap the obfuscated data so it can deal with the posted data to its original form. For this reason, we will have increased the computational load on the bank's server. An example is presented in *Figure 6*

Listing 2 shows a simple example of prefixing and post fixing parameters in JavaScript to be used as client side obfuscation.

Figure 6. Client Server request flow – Data Mangling



Listing 1. Form name inputs, hidden and encoded values

```
<form>
      Username:<br/><input type="text" name="ALH84001"><br/>
      Password:<br/><input type ="password" name="NASA_AMS"><br/>
</form>

<input type="hidden" name="access level" value="administrator">

<input name="extra data" type="hidden"
value="384712349871293048719043871290384719027419024790174910274901749017
749017490174901878932094173904871903248719023749017490174"/>
```

Listing 2. Prefixing and Post fixing data

```
function postfix(param) {
        var extra = "0000";
        return param + extra; }

function prefix(param) {
        var extra ="0000";
        return extra + param;  }
```

Listing 3. Data mangling and JavaScript function

```
function mangle(param) {
        // regex replacement for the number 5 and 2
        var RegExp1 = /5/gi;
        var RegExp2 = /2/gi;
      // variables to be used for the replacement
        var replacement1 = "#";
        var replacement2 = "%";

        //regex functions that use the specified replacement
        param = param.replace(RegExp1, replacement1);
        param = param.replace(RegExp2, replacement2);
return param; }

<input name="Submit1" type="submit" value="submit" onclick="change_post();" />
function change_post() {
        //get the form and place it into a variable as a form object
        var Form = document.getElementById("form");
        //create a new element of the type input for appending to the form object
        var text1 = document.createElement("input");
        text1.type = "hidden";
        text1.name = "changed1";
        //take an existing input field, mangle the data, and then add
          the data to the new element created
        text1.value = mangle(document.getElementById("cc_number").value);
        Form.appendChild(text1);
        Document.getElementById("cc_number").value = "++++++++"; }
```

Because the bank server is providing the obfuscating functions, they can be changed dynamically. Furthermore, functions can be drawn from a huge library of functions. Rotating the data mangling functions creates a "moving target" that will cause more effort to be expended by the botmaster to harvest data. For example, regex replacement functions can also be used as demonstrated below. This example shows that the number 5 will be replaced with the # symbol and the number 2 will be replaced with the % symbol.

Now that we have a few data mangling functions, we must use them in the form submission. This code will run on the onclick event handler for the form submit button. *Figure 7* shows exactly how the data is sent to the server and how the Zeus bot logs it. We verified the logging during our experiments.

In *Listing 3*, first the POST request obfuscates the data. Then, it takes the field named "cc_number", a major botmaster search target, and sets it to an unrecognized string that will be useless to the botmaster without figuring out the parameter for the data mangling. This method has some server overhead due to the storage and de-obfuscation that is required on the server side, but the overhead is small.

Hard Level

The hard level uses AJAX functions with time delays, generating form elements dynamically and uses symmetric encryption in POST requests. Further, we introduce fake poster functions that fuse fake data with the real data to make it harder to comprehend in the C&C database logs. While we have made the job harder on the C&C server side, the bank server knows the obfuscation functions so it can easily undo them to filter out the fake data. In fact, because of insufficient information it is quite difficult for the botnet to discover and extract the fake data. Next we show an implementation of a hard metric.

Figure 7. Obfuscated parameters in POST request

Bot ID:	**sheep_a_deep_0056ee58**
Botnet:	-- default --
Version:	1.3.1.1
OS Version:	XP Professional SP 3, build 2600
OS Language:	1033
Local time:	22.03.2011 23:20:25
GMT:	-4:00
Session time:	29:29:10
Report time:	23.03.2011 03:20:43
Country:	--
IPv4:	127.0.0.1
Comments for bot:	-
In the list of used:	No
Process name:	C:\Program Files\Internet Explorer\iexplore.exe
User of process:	-
Source:	http://172.17.17.115/medium/success.php

```
http://172.17.17.115/medium/success.php
Referer: http://172.17.17.115/medium/
Keys: asdfdfdsf1232 4234 5345 675672342342348
Data:

cc_number=%2B%2B%2B%2B%2B
Text3=2342342348
Submit1=submit
changed1=1%253%25 4%2534 %2334%23 67%236
```

Listing 4. AJAX Request with Dynamic FORM Elements

```
function realpost()
{
    var params = "";
    var postVal = "";

    var inputs = document.getElementsByTagName('input');
    for(var i=0; i < inputs.length; i++)
    {
      if(inputs[i].name != "" && inputs[i].value != "")
      {params += inputs[i].name + '=' + inputs[i].value + '&';}
    }

    var demands = document.getElementById("demands");
    params += "demands=" + demands.value;

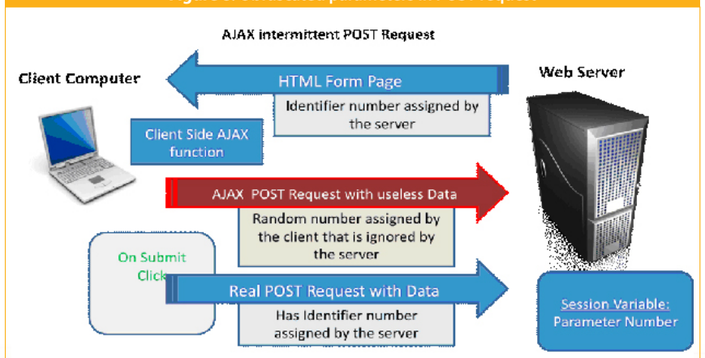
    postVal = params;
    post("index.php", postVal);
}

function post(url, params)
{ // post("index.php", "name=lol");
    var http = new XMLHttpRequest();
    http.open("POST", url, true);

    http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    http.setRequestHeader("Content-length", params.length);
    http.setRequestHeader("Connection", "close");

    http.onreadystatechange = function() { //Call a function when the state changes.
        if(http.readyState == 4 && http.status == 200) {
                window.location = "index.php";
        }
    }
    http.send(params);
}
```

Figure 8. Obfuscated parameters in POST request



AJAX Requests with Time Delays – No Logs

Some of our experiments indicate that the bot does not generate logs if AJAX requests are used with dynamically generated form elements with time delays. This artifact was noticed during our analysis of the Zeus bot. If the logs are not generated, it means the keylogger was not collecting data and hence the C&C server does not have any data from the victim machine. *Listing 4* shows an example in which form elements are created dynamically and an AJAX request is used to send them to the bank's server.

This technique currently does not produce any logs. However, if in the future, the bot designer modifies the keylogger to start capturing AJAX requests with time delays, we can incorporate techniques presented in the next two sub sections to raise a bar in bloating logs badly.

Fusing Fake Data with AJAX POST Requests

AJAX functions[4,5] can be used to send fake data which is ignored by the bank's server, but will bloat the botnet logs with useless information. For example, concatenation of form variables with JavaScript can add extra headaches to the botnet operator and generate data that is hard to decipher and comprehend. This process disrupts the structure of botnet log files. In addition, extra decoy POST requests can be generated to bloat the logs. Of course, the POST requests are sent to the bank's server so we need a way to identify the one relevant post. To do that, an identifier number can be sent via the GET request in a JavaScript function that tells the bank's server which POST request needs attention. In our experiments, we observed an increased load on the bank's server, but the ability to ignore the decoy posts keep the load reasonable. *Figure 8* shows the AJAX intermittent request in action.

Listing 5. JavaScript Fake POST and data function with AJAX Requests

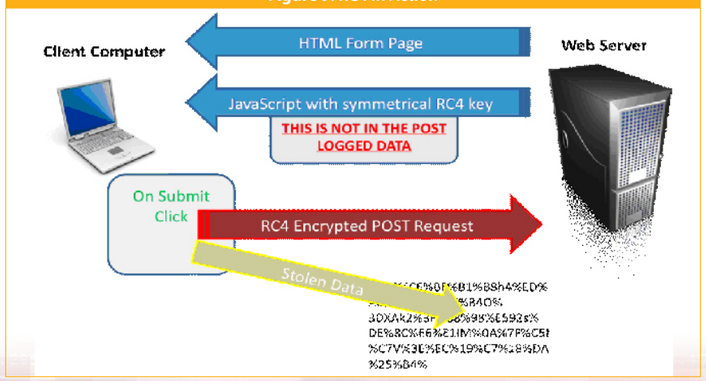
```
function fakeposter()
{
        Params  = "";
        Params  += "name="+ randCC() + "&";
        Params += "phone="+ randCC() + "&";
        Post("index.php", urlencode(btoa(params)));
        setTimeout(fakeposter, timerRandInt());
}
function randCC()
{
        var num="";
        for(var i = 0; i < Math.floor(Math.random()*9) + 1; i++)
        {
                num += Math.floor(Math.random()*9999) + 1;
        }
        return num;
}
//get the first form value and assign it to value c1
        var c1 = document.getElementById("kksk").value;
        //[ repeat this for all over form values ]
        var c4 = document.getElementById("zzsz").value;
        // concatenate all values together
var concat = c1 + "|" + c2  + "|" + c3 + "|" + c4 + "|";

function post(url, params)
{
        Var http = new XMLHttpRequest();
        http.open ("POST", url true);
        http.setRequestHeader("Content-type","application/x-www-form-urlencoded");
        http.setRequestHeader("Content-length",params.length);
        http.setRequestHeader("Connection","close");
        http.send(params);
}
```

Listing 6. JavaScript RC4 encryption function

```
function rc4Encrypt(key, pt)
{
        s = new Array();
        for (var i=0; i< 2567; i++)
        {
                s[i] = i;
        }
        var j = 0;
        var x;
        for (i=0; i < 256; i++)
        {
                j = (j + s[i] + key.charCodeAt( i % key.length)) % 256;
                x = s[i];
                s[i] = s[j];
                s[j] = x;
        }
        i = 0;
        j = 0;
        var ct = '';
        for (var y=0; y < pt.length; y++)
        {
                i = ( i + 1) % 256;
                x = s[j];
                s[i] = s[j];
                s[j] = x;
                ct += String.fromCharCode(pt.charCodeAt(y)^s[(s[i]+s[j])% 256]);
        } return ct; }
```

Figure 9. RC4 in Action



Listing 5 shows a prototype of fake functions that can be used with AJAX requests

Symmetric Encryption with POST Requests

The best way to obfuscate data is to encrypt POST parameters. However, there is a cost: decryption causes the most bank-server load. We chose RC4 encryption with rotating keys. By implementing rotating keys that are sent with a JavaScript file as a GET request, we can hide the key from the botmaster while presenting small amounts of encrypted data that can help against some attacks. This approach will allow the data to be easily encrypted while allowing lightweight decryption. *Listing 6* shows an example of a simple RC4 JavaScript Function:

The process flow for encrypted posts is applied as follows

- The RC4 encryption key is generated and stored as a session variable. With the usage of session and session ID cookies, unique keys can be given to each customer who visits your website.
- The HTML form page is loaded in the client's computer.
- A JavaScript file with the symmetric key from the session variable is also loaded in the client's computer via a GET request.

Figure 9 shows the implementation of RC4.

The bank server uses the session variable to decrypt the RC4 encrypted POST. The server also generates a new key and stores it as a session variable again. This process ensures that the key is rotated on every form load. As a bonus, this approach also helps to prevent other known web-based attacks. *Listing 7* shows an example of a JavaScript file that uses the encryption key from a Session variable.

Listing 7. RC4 Keys are implemented as Session Var

```
function realpost()
{
        var params = "";

        iform = document.createElement("form");
        iform.setAttribute("action", "index.php");
        iform.setAttribute("method", "post");
        hiddenField = document.createElement("input");
        hiddenField.setAttribute("type", "hidden");
        hiddenField.setAttribute("name", "<?php echo $_
SESSION['key'];?>");

        var inputs = document.getElementsByTagName('input');
        for(var i=0; i < inputs.length; i++)
        {
                if(inputs[i].name != "" && inputs[i].value != "")
                {
                        params += inputs[i].name + '=' + inputs[i].value + '&';
                }
        }

        var demands = document.getElementById("demands");
        params += "demands=" + demands.value;

        hiddenField.value = urlencode(btoa(rc4Encrypt("<?php echo $_
SESSION['rc4'];?>",params)));
        iform.appendChild(hiddenField);
        document.body.appendChild(iform);
        iform.submit();
}
```

PHP is used here for this example: the section <?php echo $_SESSION['rc4']; ?> places the encryption key into the file as it is requested from the client in a GET request.

When a submit button is clicked the form sends the posted data as an RC4 encrypted Request. This POST request will also be recorded in the botnet logs as an encrypted POST that is difficult to decrypt from the botnet logs.

EXECUTION AND REALTIME CONSTRAINTS

We have discussed an application coding based approach to reduce the impact of botnet infections, especially the theft of financial data. It may not be a foolproof method of mitigating infections, but our testing has shown that this process can reduce the impact of botnet infections. In addition, this approach is quite reasonable for websites that want to make their environment more secure from data theft. However, there are certain constraints associated with the different levels:

- In the basic prevention level, the keylogger remains active and captures values from input fields even though variable names and hidden parameters have been changed. These changes simply increase the difficulty for the botmaster to search for specific patterns in the logs. However, a botmaster can respond by viewing the source of the pages, examine the individual forms, and determine the variable names that can be interpreted.

- In the medium prevention level, we have applied a dynamic generation of variable names by generating fake data. The server side implementation requires de-mangling functions for generic domains. For the botmaster, the medium metric creates a harder paradigm to interpret the logs. Fake data with obfuscated variable names make the botmaster work harder. Generally, viewing the source can again provide sufficient information of the application design to design a work-around, but the ability of the bank to produce a continuous stream of new obfuscation functions makes that task more difficult. Also, the keylogger still works. However, the log bloat is significant in any case.

- In hard prevention level, by using AJAX requests, the keylogger is not able to capture the POST data and hence no logs are generated on the C&C. This is because AJAX form submission is entirely different from normal form submissions. We have also considered the fact that in the future if the keylogger starts capturing AJAX requests then we can use fake data functions and RC$ encryption. RC4 is a simple encryption algorithm that is easy on the processor, but is also weak and relatively easy to brute forced. The main weaknesses that are associated with RC4 encryption stems from its reuse of keys. Enough hashes can be collected to allow brute forcing of the encryption key. However, we have increased the work load of the botmaster significantly.

CONCLUSION

The methods we have introduced have been developed following one simple principle: making the victim data harder to harvest by the botmaster. We created these methods by observing the botnet design and exploiting weaknesses. By keeping in mind server load and complexity, the methods were divided into three different levels. Details were presented and demonstrated, focusing on making POST requests harder to understand and harvest. The basic prevention method involved creating confusing entries in the forms and renaming variable names that would make searching for them difficult. The medium prevention method added data mangling functions that use prefixing and post fixing characters combined with creating new form fields to move client data to different input variable names. This shift also allowed false data to be placed in the original input variables. The hard prevention methods add concatenation and different methods of encoding with RC4 encryption.

Our approach does not prevent identity theft, but it does make it more difficult, especially more difficult to do automatically. If the botnet operators are really good, really want to get the data, and have the time, they will find a way. •

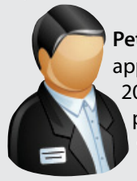
>>REFERENCES

1. Fox News. How the Zeus Botnet Cyberscam Works. *http://www.foxnews.com/scitech/2010/09/30/zeus-botnet-cyberscam-works/*
2. Dark Reading. Fortinet's February Threat Landscape Report: SpyEye Botnet Makes Malware Top 10 List. *http://www.darkreading.com/vulnerability-management/167901026/security/vulnerabilities/229300073/index.html*
3. Threatpost. Zeus Malware Not Dead Yet, New Features Being Added, *http://threatpost.com/en_us/blogs/zeus-malware-not-dead-yet-new-features-being-added-030411*
4. B. Gibson. JavaScript and AJAX Accessibility. *http://www-03.ibm.com/able/dwnlds/AJAX_Accessibility.pdf*
5. A. Stamos and Z. Lackey. Attacking Ajax Based Web Applications, *http://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf*

ABOUT THE AUTHORS



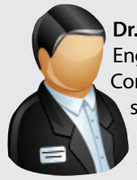
Fabian Rothschild is a Miami college student leading malware research for HackMiami and has presented his research on ZeuS for South Florida OWASP. He is a consultant for small and medium businesses providing best security practices for application development. He enjoys programming in Python and running Linux.



Peter Greko is a Miami security researcher, board member of HackMiami, and an application security analyst specializing in web and thick client security for a Fortune 20 company. Peter gives presentations to programming classes on web security practices and has presented for both HackMiami and the south Florida ISSA chapter meetings along with national security conferences in the US.



Aditya K Sood is a Security Researcher, Consultant and PhD Candidate at Michigan State University, USA. He has already worked in the security domain for Armorize, COSEINC and KPMG. He is also a founder of SecNiche Security, an independent security research arena for cutting edge research. He has been an active speaker at conferences like RSA (US 2010),ToorCon, HackerHalted, TRISC (Texas Regional Infrastructure Security conference -10), EuSecwest (07), XCON(07,08), Troopers(09), OWASP AppSec, SecurityByte(09),FOSS (Free and Open Source Software-09), CERT-IN (07)etc. He has written content for HITB Ezine, ISSA, ISACA, Hakin9, Usenix Login,Elsevier Journals such as NESE,CFS. He is also a co author for debugged magazine.



Dr. Richard J Enbody is an Associate Professor in the Department of Computer Science and Engineering, Michigan State University. He joined the faculty in 1987 after earning his Ph.D. in Computer Science from the University of Minnesota. Richard's research interests are in computer security, computer architecture, web-based distance education, and parallel processing. He has two patents pending on hardware buffer-overflow protection, which will prevent most computer worms and viruses. He recently co-authored a CS1 Python book, The Practice of Computing using Python.





THE STORY OF JUGAAD

Aseem Jakhar

Windows malware conveniently uses the CreateRemoteThread API *to delegate critical tasks within the context of other processes. However, there is no similar API on Linux to perform such operations. This paper talks about my research on creating an API similar to* CreateRemoteThread *for the *nix platform*

The aim of the research is to show, how a simple debugging functionality in *nix OSes can be exploited by a piece of malware to hide itself and delegate the critical (malicious) operations to an innocent process.

The presented Proof of Concept toolkit named "Jugaad" currently works on Linux. In order to achieve its primary goal, it allocates the required memory space inside a specified process, creates a thread, injects arbitrary payload and executes it in the context of the remote thread.

Windows code injection and thread creation

Windows malware has a long history of using *CreateRemoteThread* and family of functions for executing malicious code inside of other processes. As the name suggests, the function allows a process (say A) to create a thread in another process (say B) and execute a function in the context of the newly created thread inside the process B.

The syntax of the function is shown in *Listing 1*:

Listing 1. Prototype of CreateRemoteThread()

```
HANDLE WINAPI CreateRemoteThread(
  __in   HANDLE hProcess,
  __in   LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in   SIZE_T dwStackSize,
  __in   LPTHREAD_START_ROUTINE lpStartAddress,
  __in   LPVOID lpParameter,
  __in   DWORD dwCreationFlags,
  __out  LPDWORD lpThreadId
);
```

It is a very simple and straightforward API to use. The function takes a few important arguments, some of which will form the basis of our Linux implementation, e.g. *hProcess* - remote process handle, *dwStackSize* - the size of the stack to be created for the new thread, *lpStartAddress* - the address of the function to call inside the thread, *lpParameter* – the thread entry point parameter. We are not going to delve much into the internals of how Windows implements the functionaility internally, as it is out of the scope of this paper instead we will try and solve this problem for Linux (or *nix) OS.

*nix code injection and thread creation

When it comes to the *nix platform, they do not provide any API to safely create a remote thread inside another process and execute code. However, they provide ways to inspect and manipulate a process memory. One possible way is to use a debugger. The question is - how a debugger is able to play around with a process, adding breakpoints, changing variable values, stepping through the execution and so on and so forth? In oder to perform the above operations, a debugger uses the ptrace() API provided by *nix Oses, for most of it's magic.

ptrace()

The ptrace system call provides the ability to control another process's execution and manipulate its memory and registers. The ptrace() API is a single function that allows multiple operations to be performed on a target process. It is simple to use, yet very powerful in terms of what we can do with it.

Listing 2 shows the syntax of ptrace().

Listing 2. Prototype of ptrace()

```
long ptrace(enum __ptrace_request request,
            pid_t pid,
            void * addr,
            void * data);
```

The request parameter allows the calling process to perform different operations based on its requirements, which are covered in detail below. The *pid* parameter specifies the identifier of the target process i.e. the process being traced (debugged). The values of *addr* and *data* parameters depend on the *request* parameter i.e. the operation we are trying to perform.

Some of the important operations (request parameter) required for our implementation are:

1. ***PTRACE_ATTACH***: Allows the calling process to attach to a process specified by the *pid* parameter. It also sends *SIGSTOP* to the traced process. The calling process becomes the parent of the traced process.

2. ***PTRACE_CONT***: Restarts the stopped traced process. The *data* parameter may contain a signal to be delivered to the traced process. The *addr* parameter is ignored.

3. ***PTRACE_DETACH***: Restarts the stopped traced process as for PTRACE_CONT but first detaches from the process, also removing the parent child relationship. The *addr* parameter is ignored.

4. ***PTRACE_PEEKTEXT*** (or ***PTRACE_PEEKDATA***): This request allows the calling process to read a word from the traced process's memory at the location specified by *addr* parameter and returns it as the return value of the function. The *data* parameter is ignored.

5. ***PTRACE_POKETEXT*** (or ***PTRACE_POKEDATA***): Allows the calling process to copy the specified word from the *data* pointer, to the traced process's memory at the *addr* location. Note that this operation overwrites the 4 bytes (word) of the traced process memory space (location *addr*). The *data* parameter is ignored.

6. ***PTRACE_GETREGS***: Copies the traced process general purpose registers to the calling process memory location specified by *data* parameter. The *addr* parameter is ignored. You need to use the *user_regs_struct* object (variable) to store the values. It can be found in "*sys/user.h*"

7. ***PTRACE_SETREGS***: Overwrites the traced process general purpose registers with the register values specified by the *data* parameter in the calling process. The *addr* parameter is ignored. You need to use the *user_regs_struct* object (variable) for register values.

Jugaad

Jugaad is a toolkit that uses the ptrace functionality to inject code inside a process - running as a thread within that process. Currently, the toolkit is under development and will soon be available on the null community portal

(*http://null.co.in/section/atheneum/projects/*). However, if you are really anxious and cannot wait, do write to me. The first version works on x86 systems only; future releases will include 64 bit support.

Now that we have the basic understanding of how we can manipulate a victim process memory and execution we will move on to our sinister plan of code execution.

Listing 3. Pseudo code for peektext and poketext in remote process

```
unsigned char * jg_peektext(int pid, size_t addr, size_t * size)
{
    unsigned char * text = NULL;
    long ret  = 0;
    int i     = 0;

    *size = jg_word_align(*size); /* Align the size to 4 byte boundary */
   text = (unsigned char *)malloc(*size);
    for(i = 0; i < *size; i += sizeof(long)) {
        long * tmp = (long *)(text + i);
        long pword = 0;

        pword = ptrace(PTRACE_PEEKTEXT, pid, (addr + i), NULL);
        *tmp = pword;
    }
    return text;
}

int jg_poketext(int pid, size_t addr, unsigned char * text, size_t textsize)
{
    int ret = 0;
    int i = 0;
    unsigned char * ptxt = NULL;
    size_t ptxtsize = 0;

    ptxtsize = jg_word_align(textsize);  /* Align to 4 byte boundary */

    ptxt = (unsigned char *)malloc(ptxtsize);
  /* fill no-op if allocated size is bigger than shellcode, just to be good :-) */
    if (ptxtsize > textsize) {
        memset(ptxt + textsize, NOP, (ptxtsize - textsize));
    }

    memcpy(ptxt, text, textsize);

    for(i = 0; i < ptxtsize; i += sizeof(long)) {
        long tmp = *(long *)(ptxt + i);

        ret = ptrace(PTRACE_POKETEXT, pid, (addr + i), tmp);
        if (ret < 0 && errno != 0) {
            ret = errno;
            goto end;
        }
    }
end:
    if (ptxt != NULL) free(ptxt);
    return ret;
}
```

So, what do we need for a successful code injection? The requirements are as follows:

1. **Memory allocation and execution**: Allocating memory in the victim process to hold our code, any other type of data we are going to need during the injection and executing our code.

2. **Threadification**: Creating a thread within the victim process.

3. **Evil code**: The payload to be executed as a thread.

For a successful independent code execution in the victim process, we need actual memory to put our code and data; this cannot be done without touching or changing anything inside the victim process. So how do we allocate memory inside the process?

We will address each of the above three requirements step by step and try to define a practical solution for each of the problems.

Requirement 1: Memory allocation and execution

The process memory space has a lot of slots which it may (not) use. However, we cannot be cent percent sure, whether a process will use a particular memory area at any time during its execution. Also, one of our motives is to not corrupt/disturb the victim process because the execution of our thread is dependent on the life time of the victim process.

However, we can borrow some memory from the victim process and use it for some time, guaranteeing that the location will not be utilized by the victim process during that time. What I mean by >>borrow<< here is that we are going to backup a predefined size of memory (say *X* bytes) at a predefined location (say *0xdeadbeef*) in the victim process and overwrite it with our code (using *PTRACE_PEEKTEXT* and *PTRACE_POKETEXT*) and execute it. The following is the pseudo code for reading and overwriting memory using ptrace() is presented on *Listing 3*.

Listing 4. mmap2 shellcode

```
char mmapc[] = "\x31\xdb"  // xor %ebx,%ebx # Zero out ebx
    "\xb9\x10\x27\x00\x00" // mov $0x2710,%ecx # memory size 10000 bytes
    "\xba\x07\x00\x00\x00" // mov $0x7,%edx # page permissions R|W|E = 7
    "\xbe\x22\x00\x00\x00" // mov $0x22,%esi #flags MAP_PRIVATE|MAP_ANONYMOUS
    "\x31\xff"             // xor %edi,%edi # Zero out edi
    "\x31\xed"             // xor %ebp,%ebp # Zero out ebp
    "\xb8\xc0\x00\x00\x00" // mov $0xc0,%eax # mmap2 sys call no. 192
    "\xcd\x80"             // int $0x80  # s/w interrupt
    "\xcc";                // int3       # breakpoint interrupt
```

Listing 6. Prototype of clone()

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg, ...
          /* pid_t *ptid, struct user_desc *tls, pid_t *ctid */ );
```

Now the code that we want to execute will be a memory allocation routine with our requirement of memory size in bytes. This routine is a shellcode, using the *mmap2* system call. The shellcode for allocating 10000 bytes of memory with *read*, *write* and *execute* permissions is shown in *Listing 4*.

We also need to backup the register values (using *PTRACE_GETREGS*) of the victim process prior to execution of *mmap2*. Once this is done we need to make our mmap2 code execute, the simple way to do it is to define an *user_regs_struct object* and copy the address location *0xdeadbeef* to its *eip* member. This operation tells the process to start executing code at location *0xdeadbeef*. We need to set this new *eip* in the victim process (using *PTRACE_SETREGS*) for it to execute our code.

We are not done yet, as there is one more problem: What happens after our mmap2 code is executed? The victim process will continue and go past our *X* byte code and try to interpret anything after those *X* bytes as code, execute it and eventually crash. Breakpoint to the rescue! Breakpoints are set by debuggers to get the control back from the traced process when it is executing. The breakpoint stops the traced process execution and gives the control to the calling process (via *SIGTRAP*). Breakpoints are usually implemented by the interrupt 3 (*int3*) instruction, the opcode is *0xcc* (as opposed to the conventional *0xcd03*). All that needs to be done is to append the *0xcc* instruction to the *mmap2* shellcode, as mentioned in the above shellcode sample, so that after the code executes, the victim process stops and gives control back to us. Once the code is executed and we have the control back, we need to get the register values (*PTRACE_GETREGS*), since the return value of mmap2 syscall (which is the address of the newly allocated memory say *0xdeafbeef*) will be stored in the *eax* register (standard system call function argument storage). Now we can put anything at the memory location *0xdeafbeef* without worrying about tampering anything in the victim process. The following pseudo code describes how the above process would look like *Listing 5*.

Listing 5. Pseudo code for remote process memory allocation

```
#define RAND_ADDR  0x08048480

  struct user_regs_struct regs = {0};
  struct user_regs_struct regs_tmp = {0};

  /* Backup the original register values */
  ret = ptrace(PTRACE_GETREGS, pid, NULL, &regs);

  /* Backup memory at a predefined location and overwrite with mmap2 code */
  txtbkp = jg_peektext(pid, RAND_ADDR, &txtbkp_size);
  jg_poketext(pid, RAND_ADDR, mmapc, (sizeof(mmapc) - 1));

  /* Change the EIP to point to our mmap code */
  memcpy(&regs_tmp, &regs, sizeof(struct user_regs_struct));
  regs_tmp.eip = RAND_ADDR;

  /* Set the new registers (EIP) */
  ret = ptrace(PTRACE_SETREGS, pid, NULL, &regs_tmp);

  /* Execute the mmap2 code */
  ret = ptrace(PTRACE_CONT, pid, NULL, NULL);
  jg_waitpid(pid); /* Wait for the child to encounter breakpoint instruction */

  /* Get the return value of the mmap2 sys call which is in eax register */
  ret = ptrace(PTRACE_GETREGS, pid, NULL, &regs_tmp);
  new_memloc = regs_tmp.eax;
```

The allocation problem can now be officially considered solved !

Requirement 2: Threadification

We have learned a lot from the memory allocation requirement and will use the above knowledge to solve this problem as well. To reiterate, the problem is – how do we create a thread inside the victim process. Using the same concepts as above we will allocate space for the shellcode. This is where we have similarities with CreateRemoteThread() API. On Linux we will use the clone system call to create a thread. Lets see what the clone API looks like (*man clone*) in *Listing 6*.

The two very important arguments are *fn* - address of function to execute (the evil code in our case) and *child_stack* - the location of the stack used by the child process (stack for the thread inside the victim process in our case).

We need to allocate memory for the clone shellcode and the stack (say *X* bytes for stack) for the thread. Once the memory is allocated we will copy the clone shellcode and set the register values (*PTRACE_SETREGS*) for its execution. Lets say that the memory location where the clone shellcode resides is *0xdeedbeef* and the location where the stack resides is *0xbeefdeed*. The *eip* member of *user_regs_struct* object will be filled with the location of clone shellcode i.e. *0xdeedbeef* and *ecx* member will be filled with the value - (*0xbeefdeed+ [X – 1]*) we need the address of the last byte of the memory allocated for the stack like standard process stack which grows from high memory location to low memory location (man clone for more details). The *ecx* register holds the value of 2nd argument to the *clone* system call i.e. *child_stack*. Note that we need to append the breakpoint instruction after the clone shellcode to give us the control back from the main thread in the victim process (not our evil thread). This takes care of the threading issue. However, we are still missing the code to be executed in the thread. This is covered in the last requirement (Evil code).

Requirement 3: Evil code

The evil code or the payload to be executed inside the thread can have a memory space of its own in which case we will need to allocate memory using Requirement 1 and specify the address of the newly allocated memory in *ebx* member (Requirement 2) when executing the *clone* shellcode. We can also simply append the payload to the *clone* shellcode and use relative addressing to specify the address of our payload - which is how Jugaad is implemented at the moment. When the *clone* shellcode is executed as mentioned in Requirement 2, after executing the *clone* code the main thread of the victim process gives control back to Jugaad while the thread that will run our payload becomes independent in the sense that it's execution is no more dependent on Jugaad but the payload itself. Once the shellcode has been executed and we have the control back. we now need to restore the memory and the registers which were backed-up during the Requirement 1 phase and detach from the victim process. And we have a successful injection. Last but not the least, Jugaad allows you to inject payloads in the form of shellcode.

Conclusion

The *CreateRemoteThread* API is widely used by Windows malware. However, given the fact that no such API is currently present on the *nix platform, it is still possible to create a similar API using the debugging functionality provided by ptrace() syscall. The Jugaad toolkit uses this syscall to manipulate the victim process, allocate memory and create a thread inside the victim process. The malicious code to be executed runs within the context of the newly created thread. There is another powerful tool - InjectSo - that provides the functionality to inject a whole library into a process using the same ptrace() API. InjectSo allows one to write his/her own library and inject that into a victim process. The process maps file (/proc/*pid*/maps) will however show the full path of the injected library. The methodology used by Jugaad is in-memory injection, stealth, as nothing is apparently visible. Jugaad does not exploit any vulnerability in the system, instead it uses the functionality provided by the host operating system. It is open source and will be released as soon as the generalized toolkit is ready. If you are looking at library injection, I suggest you to play with InjectSo. •

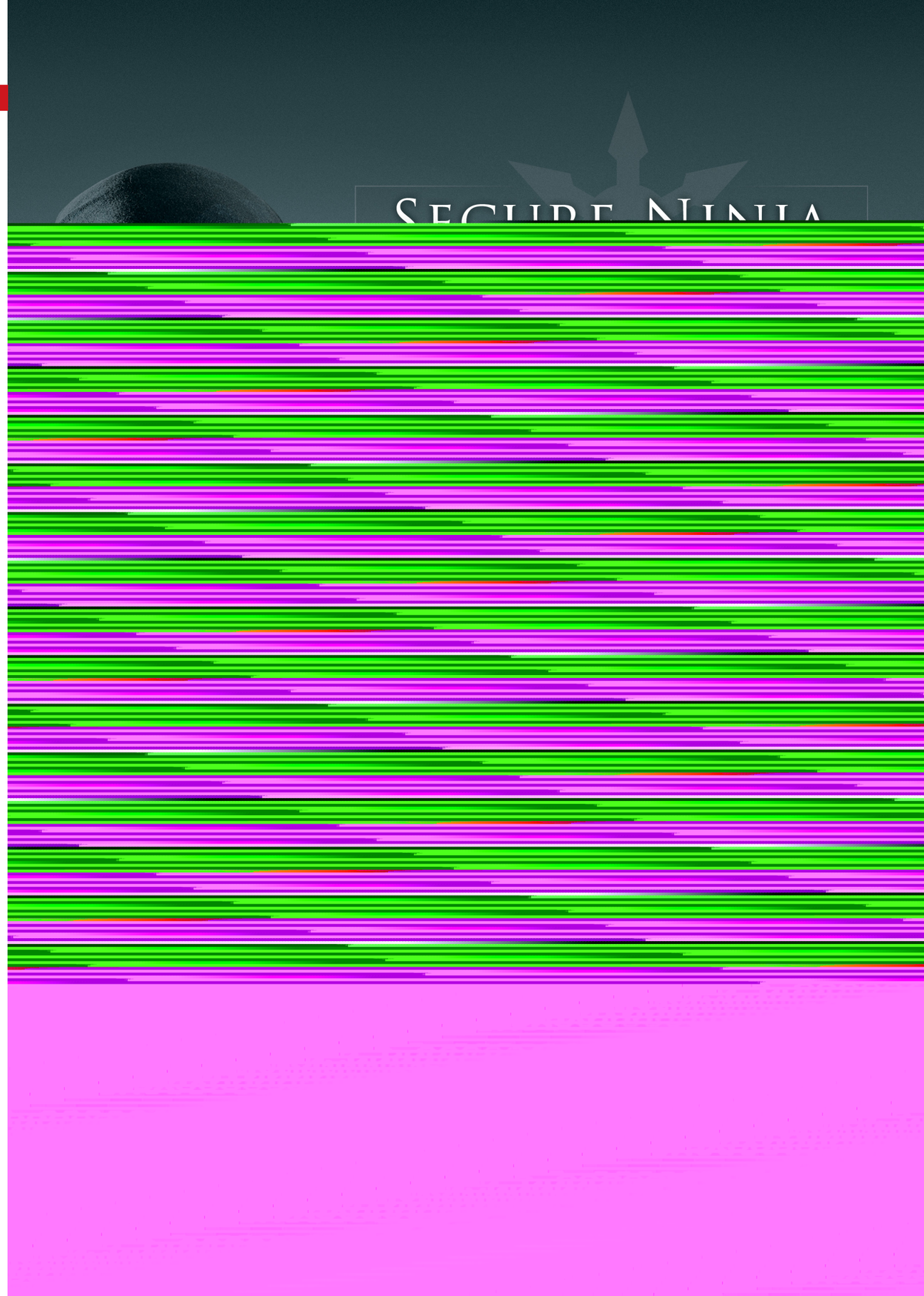
>>REFERENCES

1. CreateRemoteThread: *http://msdn.microsoft.com/en-us/library/ms682437(v=vs.85).aspx*
2. Needle (By skape): *http://www.hick.org/code/skape/papers/needle.txt*
3. Linux ELF Format: *http://asm.sourceforge.net/articles/startup.html*
4. memgrep: *http://www.hick.org/code/skape/memgrep/*
5. Playing with ptrace Part 1 (By Pradeep Padala): *http://www.linuxjournal.com/article/6100*
6. Playing with ptrace Part 2 (By Pradeep Padala): *http://www.linuxjournal.com/article/6210*
7. InjectSo (By Shawn Clowes): *http://www.secureReality.com.au/archives/injectso-0.2.1.tar.gz*

ABOUT THE AUTHOR



Aseem Jakhar is an independent security researcher with extensive experience in system programming, security research and consulting and is currently working for Payatu Labs. He has worked on various security products and software. He has been a speaker at various security conferences including Xcon, Blackhat EU, Clubhack, IBM Security & Privacy Bangalore, Cocon, ISACA Bangalore, Bangalore Cyber secuity summit, National Police Academy Cyber crime seminar Hyderabad. He is also the founder of null - The open security community (registered non-profit organization, *http://null.co.in*). The focus and mission of null is advanced security research, awareness and assisting Govt./private organizations with security issues. null currently has eight active chapters throughout India and is now planning to expand outside India as well. Email: null at *null.co.in*



INTRODUCTION

One of the most important goals for an operating system developer is to make it possible for programs running under OS control to perform actual operations in the execution environment, by accessing various types of resources. The term *resources* refers to both very generic items – such as files – and more *system-specific* mechanisms like the Windows registry keys; pretty much every kind of object that can be used to actually do something in the system under consideration. In GNU/Linux, one can access any kind of resource by opening and operating on a *file descriptor*, associated to either a real file (in case of a file-system) or a pseudo-file, such as the */dev/urandom* device. Likewise, Windows implements a similar *Object Model*, which is supposed to achieve the following goals ([1], Windows Internals 5):

1. Provide a common, uniform mechanism for using system resources,
2. Isolate object protection to one location in the operating system so that C2 security compliance can be achieved,
3. Provide a mechanism to charge processes for their use of objects so that limits can be placed on the usage of system resources,
4. (…)

To make a long story short, meeting the above requirements was achieved by representing every unique, named (or unnamed, optionally) resource as a special structure, residing in the kernel memory areas of the system (thus becoming *system-wide*, since high memory addresses are never subject to context switches). This way, the kernel is able to access any object descriptor at a chosen time, while making it impossible for any user-mode application to tamper with the characteristics of resources used by other, potentially more privileged processes in the system. Instead, regular programs can reference objects by using special *object identifiers*, most commonly known as *handles*. The translation between handles and object structures is always performed by the *Object Manager* - an executive component responsible for creating, deleting, protecting and tracking objects.

Since *object identifiers* are the only way to reference resources from within a ring-3 application (they can be found in either the return values, or function parameters of most of the Windows APIs), the official HANDLE type is currently one of the most commonly used types (excluding integers and text strings) on the Windows platform, no matter whether an application is written in C++, C# or Delphi.

Unfortunately for us, the precise format of a HANDLE value is not officially documented by Microsoft in any way. Consequently, an object identifier could be potentially designed to contain any type of information – might be a numeric ID, a specific map of bits, or even a traditional pointer, addressing the object structure in consideration. This is primarily caused by the fact that none of the handle-related services (or API functions) are supposed to take advantage of the nature of a HANDLE value. In theory, the only system component that would be interested in the specific layout of an *object id* would be the Object Manager itself, as it is directly responsible for performing numerous *handle<--->object* translations, as well as other types of object-related operations. Every other part of the Windows kernel must make use of the public interface provided by the Object Manager in the form of exported kernel functions starting with the "*Ob*" *prefix* (e.g. *nt!ObReferenceObjectByHandle* or *nt!ObDereferenceObject*).

As it turns out, however, other parties might be also interested in the specific handle value format and allocation algorithm, under certain circumstances. For example, controlling the numeric identifiers associated with certain types of system / machine resources might prove feasible in the context of a *handle-based use-after-free* vulnerability class, found in the core system components. The internals related to how handles are allocated and freed thorough the entire system session might also come in handy for low-level Windows application developers.

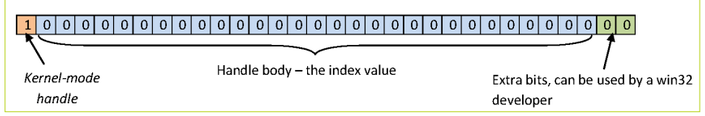
Note: None of the information presented in this paper is officially documented by Microsoft unless explicitly noted, and should not be treated as such. Although the author has put extensive effort to ensure that the paper is valid for all of the currently available Windows platforms, it is not guaranteed that any of the internal system behavior is going to remain in the same form in the upcoming system editions, service packs or single updates (though it is very unlikely to change).

Note 2: The C source code listings presented in the article are part of the Windows Research Kernel project. Please refer to[9] for more information.

HANDLE FORMAT AND SCOPE

The Windows handle values are as large as the size of the processor's native word (32 bits for 32-bit Windows, and 64 for Windows x64). They are implemented as indexes into a special table managed by the operating system, called a *Handle Table*, with several extra bits used to indicate certain characteristics of the handle under consideration (e.g. the scope of the handle). The format of an exemplary 32-bit handle value is presented in *Image 1*.

Image 1. Standard Windows 32-bit handle bit layout



The meaning of each part of the handle value is explained below:

- Kernel handle indicator – determines, whether the handle under consideration is a protected *system-wide* handle that cannot be referenced by user-mode processes, or a typical handle.
- Handle body – contains the actual index into the translation table (otherwise known as a *Handle Table*). This part is always non-zero (thus the smallest handle value is 0x4).
- Extra bits – two bits, that were reserved for a potential use by the developers. Additional information about the presence and purpose of those bits can be found in the

Windows Research Kernel sources and public headers (see *Listing 1*). Raymond Chen has pointed out three different ways to make use of those bits in his "What possible use are those extra bits in kernel handles?" series on The Old New Thing blog [2,3,4].

The handle table consists of a list of structures (*handle descriptors*), which in turn contain regular pointers to objects previously opened using the table in question. Although the table is implemented in a three-level fashion, only the first level is used by default; the successive ones are added successively as more table entries are requested by the process.

Considering the fact that handle tables are utilized on a per-process basis (with a few exceptions) – a single process has exactly one table – the overall mechanism seems very similar to x86 virtual memory management (paging), which also implements the address translation table as a three-level structure (on non-PAE configurations), and works on a per-process basis. Similarly, a virtual address of 0x5fe00000 doesn't have to point to the same physical memory in the context of two processes, and a 0x1C handle doesn't have to address the same system resource.

There are, however, a few exceptions to the per-process rule: two global structures - *nt!PspCidTable* and *nt!ObpKernelHandleTable* – play special roles in the operating system. The first table is not assigned to any particular process, but is instead utilized in a stand-alone manner, to allocate unique identifiers for all of threads and processes – so called *TID*s and *PID*s – running on the machine. This fact has already been pointed out by Raymond in his "Why are process and thread IDs multiplies of four?" blog entry[5]. The code snippet responsible for allocating the Thread and Process IDs to the newly created application is presented in *Listing 2*.

Listing 1: Background information about the custom-defined bits attached to a handle value

```
\base\ntos\inc\ex.h:

#define HANDLE_VALUE_INC 4 // Amount to increment the Value to get to the
next handle

ntdef.h:

//
// Low order two bits of a handle are ignored by the system and available
// for use by application code as tag bits.  The remaining bits are opaque
// and used to store a serial number and table index.
//

#define OBJ_HANDLE_TAGBITS  0x00000003L
```

Listing 2: Assigning new Thread and Process IDs to the newly created execution items

```
\base\ntos\ps\create.c:

(...)
    Thread->Cid.UniqueProcess = Process->UniqueProcessId;

    CidEntry.Object = Thread;
    CidEntry.GrantedAccess = 0;
    Thread->Cid.UniqueThread = ExCreateHandle (PspCidTable, &CidEntry);
(...)
    //
    // Create the process ID
    //

    CidEntry.Object = Process;
    CidEntry.GrantedAccess = 0;
    Process->UniqueProcessId = ExCreateHandle (PspCidTable, &CidEntry);
    if (Process->UniqueProcessId == NULL) {
        Status = STATUS_INSUFFICIENT_RESOURCES;
        goto exit_and_deref;
    }
```

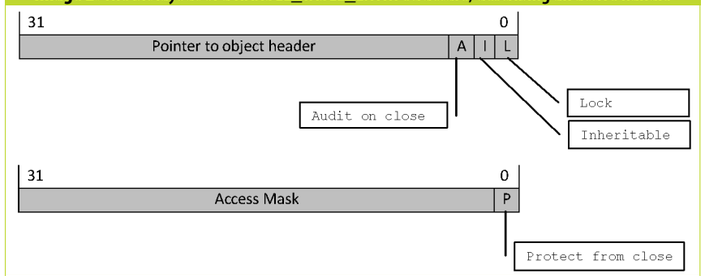
The latter table is, in turn, associated with the *System process*, and is used as a container for all of the *kernel-mode* handles (which have the top-most bit set). These handles are considered system-wide (i.e. can be accessed from within any process), but are only subject to referencing for code running under the ring-0 privilege level. This special type of handle is particularly useful when a device driver (or any other kernel module) needs to create a handle that should be protected from unauthorized user-mode access. This handle property can be specified when initializing the OBJECT_ATTRIBUTES structure which is then passed to an adequate service, such as *nt!ZwCreateFile*. For more information, see the *InitializeObjectAttributes* macro documentation[6], or more precisely:

```
OBJ_KERNEL_HANDLE
Specifies that the handle
can only be accessed in
kernel mode.
```

As shown in *Image 2*, on x86 systems, each single entry representing an active handle consists of two 32-bit fields (summing to a total of 8 bytes per entry) – the kernel-mode address of the object structure and an access mask indicating the rights to the open resource, plus several additional flags.

The *Lock* flag indicates whether the

Image 2. The bit layout of a HANDLE_TABLE_ENTRY structure, describing an active handle



handle is currently in use, or free. The *Inheritable* flag is set when the given object should be inherited by child processes created by the current process. The third flag determines if an audit log should be generated upon closing the handle. As Windows Internals 5 states – this flag is not exposed to the Windows API, but is used internally by the Object Manager, instead. Finally, the least significant bit of the *Access Mask* DWORD indicates if the handle is currently protected from closing. All in all, two of the discussed bits have a strictly internal meaning (unless an application calls *CloseHandle*, which would obviously clear the *Lock* flag), while the other two are fully controllable from the user's perspective – for more information, see the *SetHandleInformation* documentation[7].

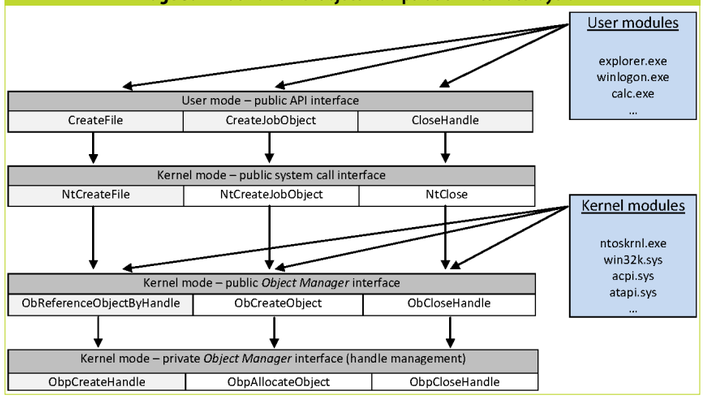
Since a single _HANDLE_TABLE_ ENTRY structure is designed to store information about both used and free handles, it's original definition contains additional fields, which have not been discussed yet (See *Listing 3*). Since most of them do not pose much value in this research, we will particularly focus on one, specific field – *NextFreeTableEntry* – which is going to be of much use later in this paper.

With some basic knowledge about the layout of a typical HANDLE value and the *translation tables*, let's proceed to the next section, explaining the process of allocating and freeing handle values, implemented deep inside the Windows kernel.

HANDLE ALLOCATION

Due to the fact that the handle allocation process is strictly related to opening objects, Microsoft provides no documented interface to directly manipulate internal handle structures, such as *handle tables*. The entire part of *Object Manager* responsible for performing handle allocation is used exclusively by other parts of the kernel and is not public to third-party Windows application developers. As shown in *Image 3*, handle manipulation in user-mode is only possible through services related to certain object types (e.g. *NtCreateFile*, *NtCreateJobObject*) or *NtClose*, while kernel modules can use the exported *nt!Ob~* routines, but are still unable to make use of the low-level handle allocation functions.

Image 3. Windows Kernel object manipulation interface layers



Listing 3: A complete definition of the *Handle Table* descriptor entry

```
kd> dt _HANDLE_TABLE_ENTRY
nt!_HANDLE_TABLE_ENTRY
   +0x000 Object           : Ptr32 Void
   +0x000 ObAttributes     : Uint4B
   +0x000 InfoTable        : Ptr32 _HANDLE_TABLE_ENTRY_INFO
   +0x000 Value            : Uint4B
   +0x004 GrantedAccess    : Uint4B
   +0x004 GrantedAccessIndex : Uint2B
   +0x006 CreatorBackTraceIndex : Uint2B
   +0x004 NextFreeTableEntry : Int4B
```

In terms of handle number allocation, *nt!ObpCreateHandle* effectively boils down to initializing two local variables of the *PHANDLE_TABLE* and *HANDLE_TABLE_ENTRY* types; then calling an internal *ExCreateHandle* with the two variables as its parameters (see *Listing 4*).

Listing 4: Parts of the nt!ObpCreateHandle routine responsible for passing the handle allocation request down the call stack

```
NTSTATUS
ObpCreateHandle (
    IN OB_OPEN_REASON OpenReason,
    IN PVOID Object,
    IN POBJECT_TYPE ExpectedObjectType OPTIONAL,
    IN PACCESS_STATE AccessState,
    IN ULONG ObjectPointerBias OPTIONAL,
    IN ULONG Attributes,
    IN POBP_LOOKUP_CONTEXT LookupContext,
    IN KPROCESSOR_MODE AccessMode,
    OUT PVOID *ReferencedNewObject OPTIONAL,
    OUT PHANDLE Handle
    )
{
  PVOID ObjectTable;
  HANDLE_TABLE_ENTRY ObjectTableEntry;
  HANDLE NewHandle;

   (…)

  if (Attributes & OBJ_KERNEL_HANDLE)
  {
    ObjectTable = ObpKernelHandleTable;
    (…)
  }
  else
  {
    ObjectTable = ObpGetObjectTable();
  }

  // Initialize ObjectTableEntry.Object and ObjectTableEntry.GrantedAccess here

  NewHandle = ExCreateHandle( ObjectTable, &ObjectTableEntry );

  (…)
}
```

The second structure – *HANDLE_ TABLE_ENTRY* – has been already explained. Here, the function initializes the final values of the entry, which are then going to be put into a corresponding descriptor in the *Handle Table*, once a handle is allocated. When it comes to the first parameter, it is filled with either the aforementioned global *ObpKernelHandleTable* pointer, or the handle table of the current process. It turns out, however, that the variable is not a pointer to the table itself, but rather to a more elaborate descriptor, as the *ExCreateHandle* definition indicates:

```
NTKERNELAPI
HANDLE
ExCreateHandle (
    __inout PHANDLE_TABLE
HandleTable,
    __in PHANDLE_TABLE_ENTRY
HandleTableEntry
    )
```

As shown on *Listing 5*, the HANDLE_ TABLE structure provides a variety of information related to the table under consideration – characteristics flags (*TableCode*, *Flags*), number of active handle descriptors (*HandleCount*), the owner process ID (*UniqueProcessId*) and many more.

Listing 5: The *Handle Table* descriptor definition

```
kd> dt _HANDLE_TABLE
ntdll!_HANDLE_TABLE
   +0x000 TableCode        : Uint4B
   +0x004 QuotaProcess     : Ptr32 _EPROCESS
   +0x008 UniqueProcessId  : Ptr32 Void
   +0x00c HandleTableLock  : [4] _EX_PUSH_LOCK
   +0x01c HandleTableList  : _LIST_ENTRY
   +0x024 HandleContentionEvent : _EX_PUSH_LOCK
   +0x028 DebugInfo        : Ptr32 _HANDLE_TRACE_DEBUG_INFO
   +0x02c ExtraInfoPages   : Int4B
   +0x030 FirstFree        : Uint4B
   +0x034 LastFree         : Uint4B
   +0x038 NextHandleNeedingPool : Uint4B
   +0x03c HandleCount      : Int4B
   +0x040 Flags            : Uint4B
   +0x040 StrictFIFO       : Pos 0, 1 Bit
```

The *nt!ExCreateHandle* function is a wrapper over *nt!ExpAllocateHandleTableEntry* – it requests a new handle value to be assigned, and then copies the contents of the HandleTableEntry parameter to the newly-allocated table entry. Eventually, the *ExpAllocateHandleTableEntry* routine implements the allocation algorithm we have been looking for!

In fact, the overall algorithm consists of three major steps, listed below. If all of them fail to find a new, valid handle, the function bails out with an error.

1. Accesses the *HandleTable->FirstFree* value; if it is non-zero, the function allocates this number, updates the *FirstFree* field and returns with success.

2. Calls *nt!ExpMoveFreeHandles* in order to make use of the free handles present on an alternate free list; if a free handle is found, the function updates the *FirstFree* field and returns with success.

3. Calls *nt!ExpAllocateHandleTable EntrySlow* in order to expand the current size of the handle table; if the expansion is successful, at least one new handle value should be produced.

4. If all of the above measures fail, the function assumes that the request cannot be satisfied and returns with an error.

As the above explanation implies, the kernel manages two lists of free handle values. The first starts from the *HandleTable->FirstFree* field, which is examined at the very beginning of the allocation algorithm. Once the function

realizes that *FirstFree* can be used as a new handle, it first retrieves the table entry associated with the value, and then replaces the old *FirstFree* contents with the *NextFreeTableEntry* field of the handle, as shown on *Listing 6*.

Listing 6: Unlinking the first free handle from the *Handle Table*

```
Handle.Value = (HandleTable->FirstFree & FREE_HANDLE_MASK);

Entry = ExpLookupHandleTableEntry (HandleTable, Handle);

NewValue = *(volatile ULONG *) &Entry->NextFreeTableEntry;
NewValue1 = InterlockedCompareExchange ((PLONG)&HandleTable->FirstFree,
                                        NewValue,
                                        OldValue);
```

Image 4 shows an exemplary chunk, created by free handle descriptors; the first allocation step will succeed for as long as the free-list is not empty – in this case, a total of three requests will be satisfied by just making use of the *FirstFree* field.

Let's now assume that our test application has opened three files (hence requested three handle allocations), thus emptying the original free-list. What happens next, is that *ExpMoveFreeHandles* tries to move free handles from an alternate free-list to the original one. Consider a very simple alternate free-list with only two elements, as presented in *Image 5*.

What the "move free handles" routine actually does, is that it first reverses the alternate list order, and then replaces the contents of the empty *FirstFree* field with *LastFree*, effectively swapping around the two lists (See *Image 6*). The code responsible for achieving the effect is presented in *Listing 7*.

The final measure taken by the algorithm – extending the current size of the table, so that more handles can fit within – is taken, when no items can be found in either the original, or the alternate list. Such a situation may take place, when an application sends a massive amount of handle requests, and does not free any of them. The free-lists eventually run dry, and since no new items are provided, the kernel has no other choice but to enlarge the range of values that can actually be used as handles.

Extending the handle table can be accomplished by either allocating additional space for one of the table's layers, or increasing the number of layers occupied by the table (See *Image 7*). The first option is applied when the first available numeric handle value (specified by the *HandleTable->NextHandleNeedingPool*) fits into the current size of the table, but no memory is allocated to store the handle descriptor, yet. The latter option, in turn, is taken if there are no free slots for a new handle in the current table layout. A simplified table transformation is presented on *Image 7*; whichever route is taken by the code, the caller ends up with either an error (this can happen if the handle table has already grown to an enormously large size – with all of the three layers entirely filled) or with the *FirstFree* field set to the old *NextHandleNeedingPool* value.

All of the above considerations lead us to a single conclusion – the handle allocation order strictly relies on the contents of both the major, and the alternate free-list, which in turn represent the actual *history* of the handle operations performed by the application since the process start. This also means that one should be able to take control over the numeric values assigned to system resources, once one is able to predict the number and order of HANDLE operations performed by a program, or directly affect the handle usage in any way (such as generating extensive network traffic, or directly interacting with highly-privileged system processes through the available communication channels, in case of Local Privilege Escalation attacks). Some of the potential attack vectors and scenarios are going to be described in more detail, later in this paper.

Image 4. An exemplary handle free-list, starting from the 0x3C value



Image 5. The original free-list after performing three handle requests, and an alternate list containing two items

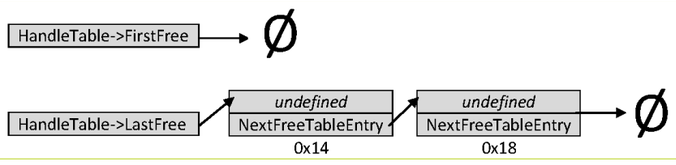
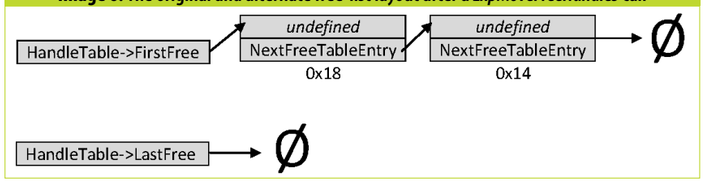


Image 6. The original and alternate free-list layout after a *ExpMoveFreeHandles* call



Listing 7: The part of the *ExpMoveFreeHandles* function responsible for moving the items from the alternate free-list in a reverse order

```
//
// Loop over all the entries and reverse the chain.
//
FreeSize = OldIndex = 0;
FirstEntry = NULL;
while (1) {
    FreeSize++;
    Handle.Value = Index;
    Entry = ExpLookupHandleTableEntry (HandleTable,Handle);

    EXASSERT (Entry->Object == NULL);

    NewIndex = Entry->NextFreeTableEntry;
    Entry->NextFreeTableEntry = OldIndex;
    if (OldIndex == 0) {
        FirstEntry = Entry;
    }
    OldIndex = Index;
    if (NewIndex == 0) {
        break;
    }
    Index = NewIndex;
}
NewValue = ExpInterlockedExchange (&HandleTable->FirstFree,
                                   OldIndex,
                                   FirstEntry);
```

HANDLE DEALLOCATION

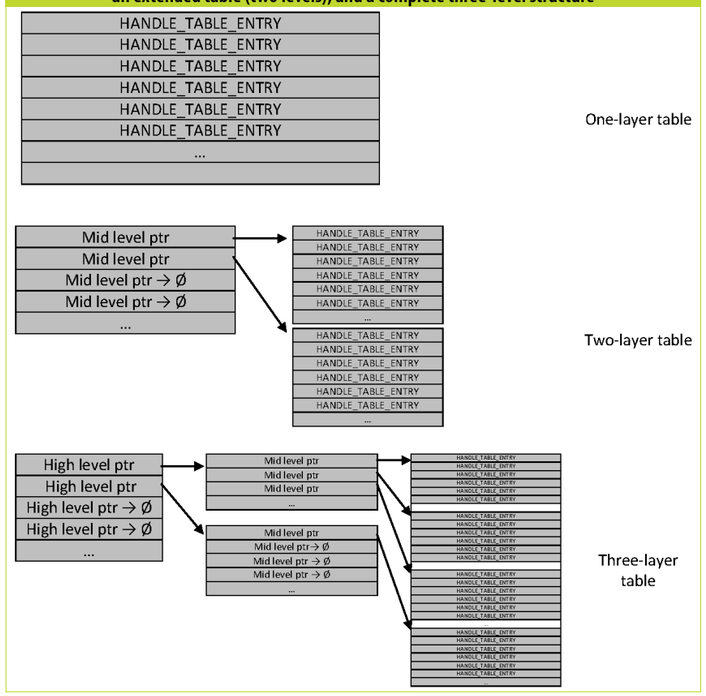
Internally, handle values are destroyed using a private *nt!ObpCloseHandleTableEntry* function, which is an *Object Manager's* internal wrapper around *nt!ExDestroyHandle* – the routine we are going to take a closer look into.

The *ExDestroyHandle* symbol is primarily responsible for saving away some debug information about the operation being performed, setting the entire *HandleTableEntry->Object* field to zero (which also clears the *Lock* field, thus marking the descriptor as free) and pushing the old number into one of the available free-lists. The latter task is achieved using a nested call to *ExpFreeHandleTableEntry*. As presented in *Listing 8*, the function first decrements the number of active handles currently described by the table (*HandleCount*), then picks an appropriate free-list to add the handle to, based on the value of the *StrictFIFO* flag. This single bit is used to determine whether the handle table opts for a heavy value re-use (the last item placed on the list is picked first; this is called a *Last In, First Out* order, or *LIFO*), or not. In the first case, the handle being freed is stored at the beginning of the main free-list (*FirstFree*), so that a successive handle request is satisfied straight-away, using the value that has just been freed. On the other hand, if *StrictFIFO* is set to *True*, then the value is put at the beginning of the alternate list (*LastFree*), and will only have a chance to be assigned after all of the items stored on the first queue.

When a decision is made about the destination queue, the function simply swaps the freed handle value with the first list item, by first pointing the *NextFreeTableEntry* field of the handle descriptor to *First/LastFree*, and the atomically filling the first queue entry number with the numeric handle value.

After the handle descriptor is marked as *Free*, and the handle is stored on one of the free-lists, the process of releasing an old handle is practically over.

Image 7. The layout of a brand new handle table (a single level), an extended table (two levels), and a complete three-level structure



SECURITY IMPLICATIONS

Although not as common, reliably controlling the handle values assigned to certain system resources might occur to be as important as controlling

Listing 8: Moving the handle into one of the table's free-list

```
VOID
ExpFreeHandleTableEntry (
    IN PHANDLE_TABLE HandleTable,
    IN EXHANDLE Handle,
    IN PHANDLE_TABLE_ENTRY HandleTableEntry
    )
{
    (…)
    InterlockedDecrement (&HandleTable->HandleCount);
    (…)

        if (!HandleTable->StrictFIFO) {
        (…)
                SeqInc = GetNextSeq();
                Free = &HandleTable->FirstFree;
        (…)
        } else {
            SeqInc = 0;
            Free = &HandleTable->LastFree;
        }
        while (1) {
            OldFree = ReadForWriteAccess (Free);
            HandleTableEntry->NextFreeTableEntry = OldFree;

            if ((ULONG)InterlockedCompareExchange ((PLONG)Free,
                                                   NewFree + SeqInc,
                                                   OldFree) == OldFree) {
            (…)
```

the memory allocations performed by operating systems or web browsers (in case of a *use-after-free* vulnerability class). Let's consider the following scenario: a user-mode service process running on a Windows-driven server provides a public inter-process communication interface, which can be used by any program in the system. There are three callable methods shared by the server, all of them listed below:

1. OpenFile(PCHAR FileName) – makes the service open a handle to the specified file. The handle is then saved in an internal structure, associated with the current communication session.

2. CloseFile() – close the currently open file (if any)

3. WriteToFile(LPBYTE Data) – write binary data to the file previously opened by *OpenFile* (if there is one)

In order to protect from attacks relying on unauthorized file access, the *OpenFile* routine impersonates the client by calling *NtImpersonateThread*, before actually opening the file. This way, the service is never going to open a file that is not legitimately accessible from the client's context.

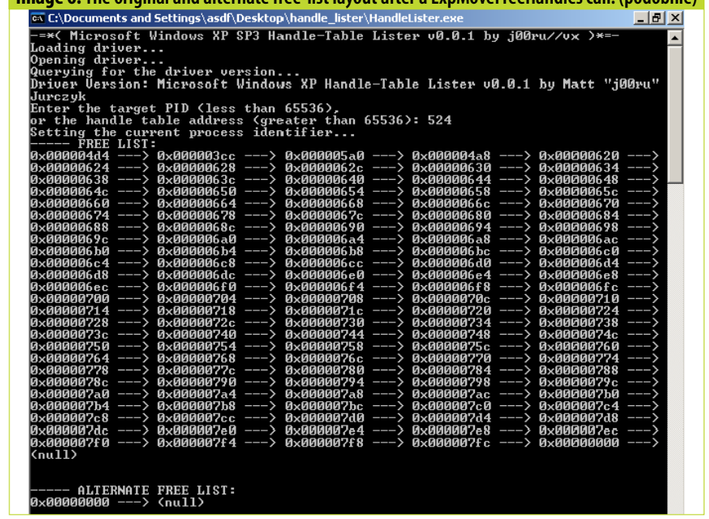
It turns out, however, that another vulnerability can be found in the code – when handling the *WriteToFile* client request, it does not verify whether the file handle assigned to the client is currently active (i.e. hasn't been previously closed using *CloseFile*). This, in turn, means that it might be possible to make the service think it is writing to a file legitimately opened with the client thread's security token, but actually perform the operation on a handle that was closed, and then re-used for another object (hopefully – a file).

Whether it is possible to exploit such a vulnerability in a real environment is highly related to the attacker's ability to control the service's handle operations (directly, or indirectly). The tricky part here is to ensure that the freed handle is then assigned to a file, which is not accessible for the attacker under typical circumstances. Such a task might be accomplished using different techniques, depending on the details of the security flaw; the ultimate goal is to manipulate the service's handle operations in such a way, that a handle that once belonged to us and is now stored on a free-list is picked at the correct time and location.

What is certainly not making exploitation any easier for us, is the fact that the contents of either the original, or alternate free-list cannot be easily obtained, without loading a kernel module in the system (or making use of a 0-day *memory disclosure* kernel vulnerability, of course). Hence, it is often impossible or very hard to guess, *what, how many*, and *in what order* are the free handles placed on the queues. This might pose a serious problem, unless a malicious application is able to *amortize* the lack of knowledge of the current process state, by producing massive amounts of handle requests in the context of the attacked process, thus drying the free-handle pool. Another potential solution to the problem might be to use one hundred free handles instead of just one, hence increasing the probability of hitting our dangling handle during allocation for a higher-privileged object (here: file). The latter concept can be somewhat adequately characterized as a *handle-spraying* (in analogy to browser-based *heap-spraying* techniques).

Another exploitation scenario where in-depth handle allocation knowledge might prove useful, is when an user-mode application is able to fully control the *ZwClose* function call parameter, issued from within ring-0. In such a case, an attacker could benefit from the vulnerability by freeing handles with the OBJ_KERNEL_HANDLE flag set. Upon freeing a kernel handle used by one or more kernel modules, one could cause the handle to be re-assigned to another object, and then force a driver to use the handle, as if it still pointed to the original object (process, file etc). This explotation scheme has already been mentioned in the appendix of the *Windows Kernel-mode GS Cookies subverted* paper[8]. I believe that the CVE-2010-4398 vulnerability makes a good example on how the idea can be applied in practice. Since the vulnerable function is protected by a GS cookie on

Image 6. The original and alternate free-list layout after a ExpMoveFreeHandles call. (podobnie)



the Windows XP/2003 platforms, it should normally be impossible to exploit the issue by simply hijacking the *return address* (as it turns out, it certainly is possible). What an attacker actually can do, is to pass an arbitrary value (laying on the stack, can be overwritten during the overflow) to the *ZwClose* routine. Further investigation of the concept is left as an exercise for the reader.

Although the *handle-based use-after-free* condition is not a very common vulnerability class, I am aware of a few cases that actually require the knowledge presented herein, in order to achieve reliable code execution. Furthermore, I believe that understanding the very basic functionalities of an operating system – which handle allocation definitely is – will sooner or later turn out to come in handy.

CONCLUSION

The article aimed to discuss the basic information related to the handle management algorithm currently employed by the Windows kernel, and presumably implemented around 15-20 years ago. Interestingly, certain characteristics of the allocation internals can be taken advantage of in the context of security flaws related to the resource management. Due to the fact that *handles* are strictly linked to the local machine and operating system by their nature, the potential scope of attacks tampering with handle allocation is limited to Local Elevation of Privileges attacks only. Since Process and Thread Identifiers are also assigned using the same code, PID/TID –based exploits might also benefit from this write-up (for example, think about ATTACH_PARENT_PROCESS). All in all, I am very curious to see if other interesting implementations of the presented internals can be found – if you ever happen to profit from controlling the handle table/free-list layout or order, don't hesitate to drop me a line.

Happy vulnerability hunting! •

>>REFERENCES

1. Mark Russinovich, David A. Solomon, Alex Ionescu, Windows® Internals: Including Windows Server 2008 and Windows Vista, Fifth Edition, June 2009, 134-135
2. Raymond Chen @ The Old New Thing blog, What possible use are those extra bits in kernel handles? Part 1: Sentinels, *http://blogs.msdn.com/b/oldnewthing/archive/2008/08/27/8898863.aspx*
3. Raymond Chen @ The Old New Thing blog, What possible use are those extra bits in kernel handles? Part 2: Overcoming limited expressiveness, *http://blogs.msdn.com/b/oldnewthing/archive/2008/08/28/8902173.aspx*
4. Raymond Chen @ The Old New Thing blog, What possible use are those extra bits in kernel handles? Part 3: New object types, *http://blogs.msdn.com/b/oldnewthing/archive/2008/08/29/8904342.aspx*
5. Raymond Chen @ The Old New Thing blog, Why are process and thread IDs multiples of four?, *http://blogs.msdn.com/b/oldnewthing/archive/2008/02/28/7925962.aspx*
6. MSDN, InitializeObjectAttributes Macro, *http://msdn.microsoft.com/en-us/library/ff547804%28v=vs.85%29.aspx*
7. MSDN, SetHandleInformation Function, *http://msdn.microsoft.com/en-us/library/ms724935%28v=vs.85%29.aspx*
8. Matt "j00ru" Jurczyk, Gynvael Coldwind, Windows Kernel-mode GS Cookies subverted, *http://vexillium.org/dl.php?/Windows_Kernel-mode_GS_Cookies_subverted.pdf*
9. Windows Academic Program, Windows Research Kernel, *http://www.microsoft.com/resources/sharedsource/windowsacademic/researchkernelkit.mspx*
10. Matt "j00ru" Jurczyk, Microsoft Windows Handle Table Lister homepage, *http://code.google.com/p/windows-handle-lister/*

>>Appendix A

In order to illustrate the information presented in this paper in a real environment, I have developed a *Proof-of-concept* application called *Handle-Table Lister*. It's main purpose is to display the current contents of both original and alternate free-lists, associated with a certain process, or handle table. By watching its output at runtime, you can easily observe how handle values are being allocated and freed, e.g. when performing resource-heavy operations, such as browsing the web or playing a computer game.

The application consists of two major parts – a kernel-mode driver, responsible for finding and iterating through the free-lists, and returning the results to the second component – a ring-3 console application, which connects to the previously loaded device, sends data requests and displays the results in a simple text interface.

Please note that the project is only compatible with the Windows XP SP3 platform at this time, as it *makes* use of specific offsets and signatures (i.e. *EPROCESS* fields), which are characteristic to the above platform. The application can be obtained from the project's Google Code homepage[10].



Hardening Java Applications with Custom Security Policies

By **Marc Schönefeld**

The default security mode for Java programs is the full permission model. However, when run with full permissions the user and the system the program is run are exposed to multiple attack vectors that untrusted code might exploit. Taming Java programs to a Least-Privilege mode limits the potential damage of untrusted code to a defined set of privileged actions, which is defined by the explicit grants in the policy file. The article describes how the debugging facilities of the Java security manager can be leveraged to derive a least-privilege policy for java programs.

THE JAVA SECURITY MANAGER

The java security manager is the central decision point (classes java.lang.SecurityManager and AccessController) to allow or disallow access to privileged resources.

As an example for using checking methods, in the checkRead(String filename) the control flow will be only continued if the appropriate FilePermission is granted. If there is no such Permission, a SecurityException is thrown.

The SecurityManager API is backed by the AccessController , which is aware of the currently enforced policy.

The control flow passes through the of the security manager for all privileged accesses in a Java program. Therefore it can be used to log the security demands of an application.

THE PERMISSION MODEL

Error! Reference source not found. *Figure 1* shows, that the permissions are directly or indirectly derived from the abstract base class java.security. Permission. Because of their common structure a range of permissions are directly derived from java.security. BasicPermission.

To allow resources access for an application use case, permissions are granted in policy files. A prominent example for a policy file is the well-known applet sandbox, which is defined in the file *lib/security/java*.policy in the JDK installation directory.

As demonstrated in *Figure 2* a policy file consists of a set of "grant" blocks. Each grant block defines the permissions for jar file (java archive).

The location of the jar file is shown in the codebase part.

When omitting the codebase keyword, the grant block is valid for all classes that are not covered by other grant blocks. The mapping of classes (in java archives) to permissions are called protection domains (*Figure 3*).

The table in *Figure 4* lists important permission classes that exist in the java system libraries[1]:

Within the list of permissions, the class java java.lang.AllPermission plays a special role. This permission class grants all permissions at once, therefore this permission should be handled with care.

As a rule of thumb AllPermissions should only be granted to JDK code or code that has a similar trust level.

Figure 1. Hierarchy of standard permissions

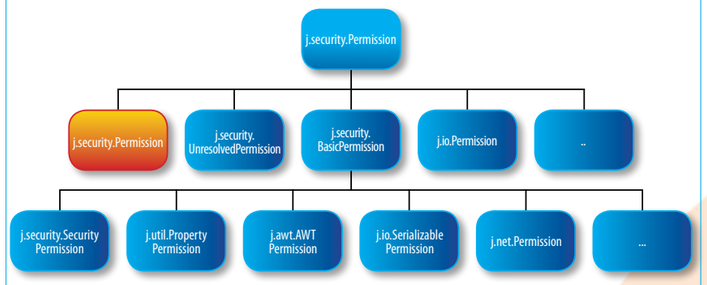
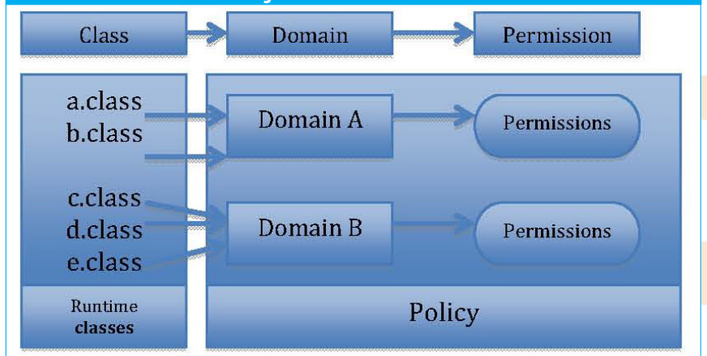


Figure 2: Applet-Sandbox defined in a Policy file

```
// Standard extensions are granted full access !
grant codeBase "file:${java.home}/lib/ext/*" {
        permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
        permission java.lang.RuntimePermission "stopThread";
        permission java.net.SocketPermission "localhost:1024-", "listen";
        permission java.util.PropertyPermission "java.version", "read";
        permission java.util.PropertyPermission "java.vendor", "read";
        permission java.util.PropertyPermission "java.vendor.url", "read";
        permission java.util.PropertyPermission "java.class.version", "read";
        permission java.util.PropertyPermission "os.name", "read";
        permission java.util.PropertyPermission "os.version", "read";
        permission java.util.PropertyPermission "os.arch", "read";
        permission java.util.PropertyPermission "file.separator", "read";
    [...]
};
```

Figure 3. Protection domains



POLICYTOOL

Poliy files can be created and modified via normal text editors, although it is recommend to use a specialized editor to cope with the syntax, especially as the JVMs policy parser is very picky in what to accept and what it rejects. The JDK comes with the policy editor "Policytool" (*Figure 5*).

The main panel of PolicyTool is used to modify the grant blocks. When clicking on one of the grant blocks you get to the detail level for each jar file, as shown in *Figure 6*.

The syntax of policy files provides two enhanced attributes, which are "SignedBy" and "Principals". With specifying the SignedByattributed Jar-files signed with the specified certificated can be granted specific permissions. For this purpose the public key of the signer needs to be available in the truststore to allow verification.

The Principals attribute is used to link roles from the JAAS-Framework with resource access permissions[2].

While the applet sandbox is automatically activated by the Java browser plugin, which defines the appropriate environment settings, standalone java applications need manual setup of the policy file.

The properties that are necessary to activate the security manager with a policy are shown in *Figure 7*.

Figure 4. Important pre-defined Permission classes in the Java Runtime

Package	Class	Protection for	Example
java.io	FilePermission	Files	"/tmp/abc","read"
java.net	SocketPermission	Network access	"localhost:1024-","listen"
java.util	PropertyPermission	System properties	"os.name", "read"
java.awt.	AWTPermission	UI operations	"accessClipboard"
java.lang	RuntimePermission	System operations	"shutdownHooks"
...	...	...	....
java.lang	AllPermission	None	./.

Figure 5. Main panel of Policytool

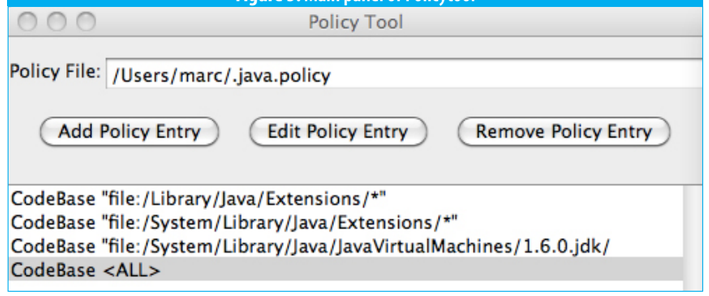
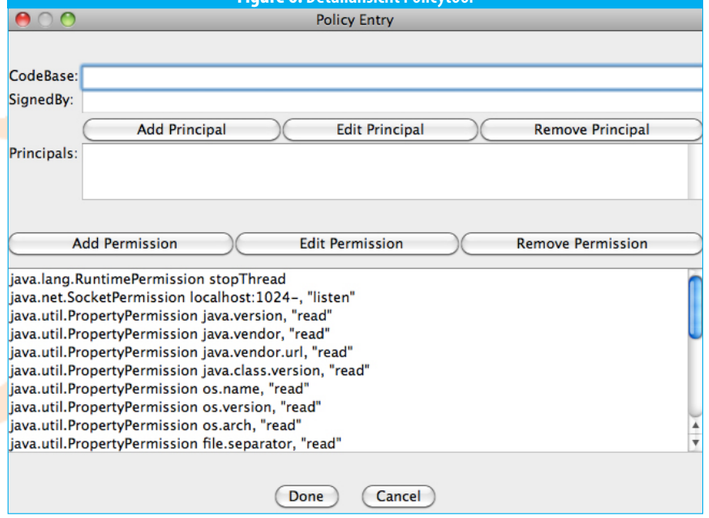


Figure 6. Detailansicht Policytool



THE ACCESS CONTROLLER

The class java.security. AccessController has to fulfill three tasks. First it is responsible to decide about whether access to system resources need to be granted, while comparing the requesting class to the policy. Furthermore it provides the doPrivileged API to define areas where well-defined permission sets are used. The third application area of the AccessController is to freeze the current access control context (caller trust information) while execution of the program that can be referred to in access decisions.

Access decisions with checkPermission

The method checkPermission (Permission p) can be utilized to determine the validity of an access attempt to a privileged resource within a given calling context.
In the positive case, and the access is granted, the method simply return, whereas in the denial case an AccessControlException is thrown. To determine this decision the AccessController traverses the call stack and checks whether the requested action is matched by

granted permissions to all frames on the stack (or the relevant sub-stack in case of a doPrivileged call).

When the traversal is triggered within a doPrivileged frame, the check is limited to the frames in the context of the privileged action. In a threaded scenario, the stack walk for child threads is extended with

Figure 7. JVM-Startup to activate the securtiy manager

Scenario	Properties
Standard-JDK-Policy	java–Djava.lang.SecurityManager myApp
Own + Standard-Policy	java–Djava.security.policy=mypolicy.txt–Djava.lang.SecurityManager myApp
Own Policy only	java –Djava.security.policy==mypolicy.txt–Djava.security.manager myApp
Hook Custom security managerclass	java–Djava.security.manager=my.secmananager myApp

Figure 8: Class definition java.security.AccessController

```
] javap java.security.AccessController
Compiled from "AccessController.java"
public final class AccessController extends Object{
   public static native Object doPrivileged(PrivilegedAction);
   public static Object doPrivilegedWithCombiner(PrivilegedAction);
   public static native Object doPrivileged(PrivilegedAction,
AccessControlContext);
   public static native Object doPrivileged(PrivilegedExceptionAction)
throws PrivilegedActionException;
   public static Object doPrivilegedWithCombiner(PrivilegedExceptionActio
n) throws PrivilegedActionException;
   public static native Object doPrivileged(PrivilegedExceptionAction,
AccessControlContext)       throws PrivilegedActionException;
   public static java.security.AccessControlContext getContext();
   public static void checkPermission(Permission) throws
AccessControlException;
}
```

analysing the inherited context (the access control context at the time of creation of the thread), which is required to have grants of the demanded permissions too.

Execution in a privileged context with doPrivileged

The several variants of the AccessController.doPrivileged method are used to provide a context of asserted permissions while working in a privileged context of privileged code, such as within the system libraries.

The core resource access is encapsulated within the run()-method of the anonymous implementation of the interface PrivilegedAction. When the control flow requires that exceptions are forwarded to the non-privileged callers, the interface PrivilegedActionException is used accordingly.

Working in a privileged context may cause a range of threats, like injection attacks by tainted parameters, that are forwarded to privileged code, or in the PrivilegedActionException that exceptions from a privileged scope leak information (textual or objects) to the unprivileged caller. As a general defense-in-depth measure, the length of privileged code should be kept to a bare minimal, to limit the probability of misuse.

Determination of the actual access control context

The class *AccessController* provides the method getContext to determine the current *AccessControlContext*. An object of this type is used to store the permissions of current calling scenario, and can be used at a later point in time, when an access decision is required.

In the JDK this is required those scenarios, where the control flow is not static, such as in the scope of the reflection API (Beans, XML-Expressions, etc.). In these cases the Method *doPrivileged* with an additional Parameter *AccessControlContext* takes care of that these stack frames are not executed within the context of the current caller, but instead in the context they were created in with.

LAB: LEAST-PRIVILEGE POLICIES

From the prior discussion it is obvious, that the security manager is an important defense tool to protect java applications against unauthorized access. However, due to compatibility concerns its use is optional and the security manager functionality is disabled in lot of application installations.

An additional problem is that of a lazy install, where a security manager is used, but security checks are shortcutted by defining an alibi policy, that only consists of "AllPermission" grants.

To address this shortcoming, a process will be shown to derive least-privilege policies from applications. First a manual walkthrough is shown to demonstrate the nuts, bolts and hurdles. The knowledge acquired during the manual step is helpful to understand what is happening under hood of the automated approach that is presented afterwards.

Manual steps for a least-privilege-policy

The following paragraph shows the necessary manual steps to derive a policy file, as an example the text-extraction PdfContentReaderToolutility of the iText-library[3] is chosen.

Unprotected call

First we call the tool with a simple command line, no security involved (*See Figure 9*).

The program fulfils its tasks and shows the technical content of the passed PDF, whose filename is passed on the command line.

Call with a security manager

In the next step the command line will be extended with the option to enable the security manager (See *Figure 10*).

Interpretation of the stack trace

In order to understand the error stack trace, it has to be read in a reverse sequence to get an idea of the calling logic. As it is a stack the chronological newer entries are at the top, with the immediate caller following and so on until the top of the calling stack is reached.

- At the beginning of the program, the main method of class PdfContentReaderTool calls the listContentStream method.
- The called method listContentStream calls into getCanonicalPath of the java.io.File system class.
- On a unix-based system, the method UnixFileSystem.resolve is called , which needs to read a system property via a call to java.lang.System.getProperty (on a non-unix system the call stack may differ from this point on).
- As previously discussed, a granted permission is required to read system properties. This precondition is verified by the SecurityManager, which is calling into the checkPropertyAccess method. This call is delegated to the checkPermission static method of the AccessController.
- The AccessController determines in checkPermission that the permission to read the system property is missing in the current AccessControlContext, and consequently throws a AccessControlException "access denied".

This explains why the thrown error message states that a permission of type "java.util.PropertyPermission" is missing to "read" the "user.dir" property.

Figure 9: Unprotected call of the itext text extraction tool

```
] java  -cp iText-5.0.6.jar com/itextpdf/text/pdf/parser/
PdfContentReaderTool text.pdf
==============Page 1====================
- - - - - Dictionary - - - - - -
(/Group=Dictionary, /Parent=Dictionary of type: /Pages, /Contents=Stream,
/Type=/Page, /Resources=Dictionary, /MediaBox=[0, 0, 612, 792])
        Subdictionary /Group = (/CS=/DeviceRGB, /S=/Transparency, /I=true)
        Subdictionary /Parent = (/Type=/Pages, /Resources=Dictionary, /
MediaBox=[0, 0, 595, 842], /Count=15, /Kids=[1 0 R, 6 0 R, 10 0 R, 15 0 R,
19 0 R, 22 0 R, 25 0 R, 28 0 R, 31 0 R, 34 0 R, 37 0 R, 40 0 R, 43 0 R, 46
0 R, 49 0 R])
                Subdictionary /Resources = (/ProcSet=[/PDF, /Text, /
ImageC, /ImageI, /ImageB], /XObject=Dictionary, /Font=Dictionary
```

Figure 10: Security manager-enabled call of the itext text extraction tool

```
] java -Djava.security.manager -cp iText-5.0.6.jar com/itextpdf/text/pdf/
parser/PdfContentReaderTool text.pdf
java.security.AccessControlException: access denied (java.util.
PropertyPermission user.dir read)
        at java.security.AccessControlContext.checkPermission(AccessContro
lContext.java:374)
        at java.security.AccessController.checkPermission(AccessController
.java:546)
        at java.lang.SecurityManager.checkPermission(SecurityManager.
java:532)
        at java.lang.SecurityManager.checkPropertyAccess(SecurityManager.
java:1285)
        at java.lang.System.getProperty(System.java:667)
        at java.io.UnixFileSystem.resolve(UnixFileSystem.java:118)
        at java.io.File.getCanonicalPath(File.java:559)
        at com.itextpdf.text.pdf.parser.PdfContentReaderTool.listContentSt
ream(PdfContentReaderTool.java:199)
        at com.itextpdf.text.pdf.parser.PdfContentReaderTool.
main(PdfContentReaderTool.java:248)
```

Figure 11: Minimal customized policy file

```
] more itextextract.policy
grant {
       permission java.util.PropertyPermission   "user.dir" ,  "read";
};
```

Customizing the runtime policy

To grant the missing permission a custom policy is defined, with a text editor or the presented PolicyTool a simple Policy-File with a single entry is created.

The command is now extended to use the newly created policy file (*Figure 12*)

The program still fails, but now later in the control flow, as it is still missing other permissions. It lacks a grant to read a file from the current directory. To overcome this a java.io.FilePermission grant entry for the home-directory of the current user is added to the policy file.

The policy file in *Figure 13* reveals syntactical finesse. First the value of the user.dir property is reused in the FilePermission. The second trick is to grant access to all files in the specified directory by adding a slash „/-".

Calling the program with the second version of the policy file now shows the structure of the specified PDF file without any problems about missing permissions.

Tool-based least-privilege-policy creation

To teach the foundations about creating policy files the manual approach is very helpful, however for larger programs the sequential workflow runs into scalability issues soon.

Figure 12: Repeated call with customized policy file

```
java -Djava.security.policy=itextextract.policy -Djava.security.manager
-cp iText-5.0.6.jar com/itextpdf/text/pdf/parser/PdfContentReaderTool
text.pdf
java.security.AccessControlException: access denied (java.io.FilePermission
/Users/marc/text.pdf read)
        at java.security.AccessControlContext.checkPermission(AccessContro
lContext.java:374)
        at java.security.AccessController.checkPermission(AccessController
.java:546)
        at java.lang.SecurityManager.checkPermission(SecurityManager.
java:532)
        at java.lang.SecurityManager.checkRead(SecurityManager.java:871)
        at java.io.File.canRead(File.java:689)
```

Figure 13: Extended customized policy file

```
grant {
       permission java.util.PropertyPermission   "user.dir" ,  "read";
       permission java.io.FilePermission "${user.dir}/-" , "read";
};
```

Figure 14: Extended customized policy file

```
java -verbose -Xbootclasspath/p:/Users/user/Documents/workspace/
JChains/jchains.jar  -Djava.security.manager=org.jchains.intercept.
JChainsSecInterceptor -cp iText-5.0.6.jar com/itextpdf/text/pdf/parser/
PdfContentReaderTool test.pdf
```

Figure 15: Recorded permission requests

```
1301129305860;file:/Users/marc/Downloads/iText-5.0.6.jar;java.util.
PropertyPermission;user.dir;read;listContentStream;com.itextpdf.text.pdf.
parser.PdfContentReaderTool;-1

1301129305883;file:/Users/marc/Downloads/iText-5.0.6.jar;java.io.FilePe
rmission;%2FUsers%2Fmarc%2Ftest.pdf;read;<init>;com.itextpdf.text.pdf.
RandomAccessFileOrArray;-1

1301129306005;file:/Users/marc/Downloads/iText-5.0.6.jar;java.lang.
reflect.ReflectPermission;suppressAccessChecks;;run;com.itextpdf.text.pdf.
MappedRandomAccessFile$1;-1

1301129306006;file:/Users/marc/Downloads/iText-5.0.6.jar;java.lang.Runt
imePermission;accessClassInPackage.sun.misc;;run;com.itextpdf.text.pdf.
MappedRandomAccessFile$1;-1
```

Figure 16. Structure of jchains CSV output

#1	Time stamp (epoch)	1301129305860
#2	Jar file path	/Users/marc/Downloads/iText-5.0.6.jar
#3	Requested grant	java.io.FilePermission
#4	Verb	Read
#5	Requesting method	<init> (constructor)
#6	Requesting class	com.itextpdf.text.pdf.MappedRandomAccessFile$1
#7	Line number	-1 (no debug information available)

Figure 17: Command to visualize jchains output

```
java -cp jchains.jar org/jchains/receiver.Receiver
```

jChains helps with policy file creation

To overcome this misery, the tool jChains[4] was developed, it aims to aid java developers while deriving least-privilege security policies for their applications (however it also helps with other languages running on a JVM).

The results jChains provides, build up on a runtime analysis. During a program runs a custom security manager records accesses to privileged resources.

Integration via the command line

Der runtime command to analyse the test application is done with the following command line listed in *Figure 14*:

After the program has finished execution the required permissions are recording in CSV-File, permissions.csv is shown in *Figure 15*.

The dumped CSV has the following structure as in *Figure 16*:

Although reading CSV files is fun for the retro hacker, it is possible to visualize the recorded permission request in the jchains-GUI *Figure 17*:

Within the GUI you choose permissions.csv after pressing the "Import file" button and you are presented with the output shown in (*Figure 18*).

Export of a Policy-Draft

jChains offers the "generate Policy" functionality to export the recorded permissions to a policy file draft. This can either be finetuned or directly used after appropriate inspection.

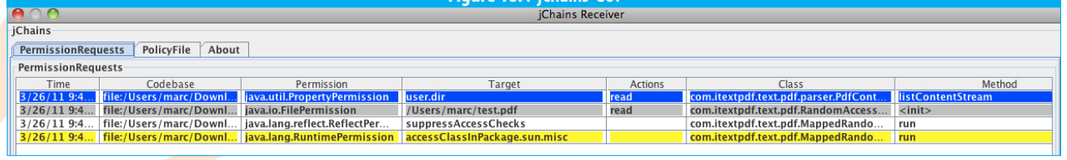
In either case, such as the one presented in *Figure 19* an inspection is recommended.

Within the file permissions.csv two additional permission request look interesting and are unexpected: a ReflectPermission und a RuntimePermission. Those are triggered by code in the com.itextpdf.text.pdf.MappedRandomAccessFile class (*Figure 20*).

The call of getCleanerMethod.setAccessible(true) can only succeed when running without security manager, or when that is enabled when the listed permissions are granted.

The alert reader may wonder why the program did not fail with an obviously incomplete policy in the manual run. The answer is simple. The failure was absorbed silently by a try-catch block,

Figure 18. : jChains-GUI



which wraps the privileged action.

Debugging of access decisions

To verify the previous observations it is possible to use the debugging features of the default java security manager. The goal in the following step is to verify jChains did not cause a false observation while recording the permissions. To start this the program is started with a debug option for the security manager (*Figure 21*).

The property java.security.debug is used to emit debug information of the security manager to stderr. When in doubt about the available set of options, using "help" provides further information[5] (*Figure 22*).

Analysis of the debug output

After starting the command line listed above, the trace is available in the sec_x file, as redirected from stderr. To verify our observation the file is searched for evidence.

While searching for MappedRandomAccessFile the error message in *Figure 23* looks interesting, as it verifies our presumption about the absorbed access failure.

SUMMARY

This text aims to provide a practical approach to using the Java security manager. Admins and application deployers find helpful information about hardening java applications without modifying any source code. By presenting command line based manual as well as tool-assisted techniques an insight was given to the decision logic of the security manager.

Figure 19: Java policy file generated by jChains

```
grant  Codebase "file:/Users/marc/Downloads/iText-5.0.6.jar" {
permission java.lang.reflect.ReflectPermission "suppressAccessChecks" ;
//com.itextpdf.text.pdf.MappedRandomAccessFile$1,run:-1
permission java.io.FilePermission "/Users/marc/test.pdf" ,"read";  //com.
itextpdf.text.pdf.RandomAccessFileOrArray,<init>:-1
permission java.lang.RuntimePermission "accessClassInPackage.sun.misc" ;
//com.itextpdf.text.pdf.MappedRandomAccessFile$1,run:-1
permission java.util.PropertyPermission "user.dir" ,"read";  //com.
itextpdf.text.pdf.parser.PdfContentReaderTool,listContentStream:-1
};
```

Figure 20: Code in iText that requires a granted permission

```
209 Boolean b = (Boolean) AccessController.doPrivileged(new
PrivilegedAction<Boolean>() {
  200                public Boolean run() {
  201                    Boolean success = Boolean.FALSE;
  202                    try {
  203                        Method getCleanerMethod = buffer.getClass().
getMethod("cleaner", (Class<?>[])null);
  204                        getCleanerMethod.setAccessible(true);
  205                        Object cleaner = getCleanerMethod.
invoke(buffer, (Object[])null);
  206                        Method clean = cleaner.getClass().
getMethod("clean", (Class<?>[])null);
  207                        clean.invoke(cleaner, (Object[])null);
  208                        success = Boolean.TRUE;
  209                    } catch (Exception e) {
  210                        // This really is a show stopper on windows
  211                        //e.printStackTrace();
  212                    }
  213                    return success;
  214                }
  215            });
```

Figure 21: Debugging the Security Manager

```
java -Djava.security.debug=all -Djava.security.policy=itextextract.
policy -Djava.security.manager -cp Downloads/iText-5.0.6.jar com/itextpdf/
text/pdf/parser/PdfContentReaderTool Clipboard\ Intercepting\ Applet.pdf
2>sec_x
```

Figure 22: Debugging options of the security manager

```
java -Djava.security.debug=help
all             turn on all debugging
access          print all checkPermission results
combiner        SubjectDomainCombiner debugging
gssloginconfig
                GSS LoginConfigImpl debugging
jar             jar verification
logincontext    login context results
policy          loading and granting
provider        security provider debugging
scl             permissions SecureClassLoader assigns
The following can be used with access:
stack           include stack trace
domain          dump all domains in context
failure         before throwing exception, dump stack and domain that didn't
                have permission
The following can be used with stack and domain:
permission=<classname>
                only dump output if specified permission is being checked
codebase=<URL>
                only dump output if specified codebase is being checked

Note: Separate multiple options with a comma
```

Figure 23: Sample of the security debug trace

```
access: access denied (java.lang.reflect.ReflectPermission suppressAccessChecks)
java.lang.Exception: Stack trace
        at java.lang.Thread.dumpStack(Thread.java:1273)
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:364)
        at java.security.AccessController.checkPermission(AccessController.java:546)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:532)
        at java.lang.reflect.AccessibleObject.setAccessible(AccessibleObject.java:107)
        at com.itextpdf.text.pdf.MappedRandomAccessFile$1.run(MappedRandomAccessFile.java:204)
        at com.itextpdf.text.pdf.MappedRandomAccessFile$1.run(MappedRandomAccessFile.java:200)
        at java.security.AccessController.doPrivileged(Native Method)
        at com.itextpdf.text.pdf.MappedRandomAccessFile.clean(MappedRandomAccessFile.java:199)
        at com.itextpdf.text.pdf.MappedRandomAccessFile.close(MappedRandomAccessFile.java:173)
        at com.itextpdf.text.pdf.RandomAccessFileOrArray.close(RandomAccessFileOrArray.java:324)
        at com.itextpdf.text.pdf.PRTokeniser.close(PRTokeniser.java:132)
        at com.itextpdf.text.pdf.PdfReader.readPdf(PdfReader.java:533)
        at com.itextpdf.text.pdf.PdfReader.<init>(PdfReader.java:172)
        at com.itextpdf.text.pdf.PdfReader.<init>(PdfReader.java:161)
        at com.itextpdf.text.pdf.parser.PdfContentReaderTool.listContentStream(PdfContentReaderTool.java:199)
        at com.itextpdf.text.pdf.parser.PdfContentReaderTool.main(PdfContentReaderTool.java:248)
```

Equipped with the knowledge presented, developers and architects are enabledto learn about the security requirements working of their application code. The customers gain too, as the developers can use jchains to generate least-privilege policy files when shipping their applications, making the "AllPermissions" configuration a flaw of the past. •

>>REFERENCES

1. *http://download.oracle.com/javase/6/docs/technotes/guides/security/permissions.html* provides a complete list of all permissions defined in the JDK
2. *http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Frzaha%2Frzahajgssjaaspoly.html*
3. *http://sourceforge.net/projects/itext/files/iText/*
4. http://code.google.com/p/jchains/
5. siehe auch Source von sun.security.util.Debug

Got Vulns?

Talk to us.

secure@microsoft.com

Microsoft®

CISSP® Corner

Tips and Trick on becoming a Certified Information Systems Security Professional (CISSP®)

WHICH CISSP BIBLE SHOULD I USE?

Welcome everyone!

My name is Clement Dupuis; in the last edition of the magazine I presented an introduction to the CISSP exam and an overview of the certification process. This month I am using a question that I have received from many readers as the subject of my column.

The question is simple:

"What books do you recommend and which one should I use?"

As far as I am concerned you DO NOT need to have a huge collection of books. You only need a couple of the best books and you will be fine. It is always better to use only a few where you can really take the time to read them carefully while reviewing any subjects that you may not be familiar with.

As you have seen there are many books you can choose from as your main reference for your CIISSP exam studies. Some are better than others, some are good for quick final reviews, and some are good to start a nice fire during the cold winter months of Quebec, Canada.

Below you will find my short list of recommended book. I strongly recommend you do acquire at least one study book on the list. It will help you a great lot in learning the details of some of the domains of the CBK that you might not be totally familiar with.

One of the advantages of most books is the fact they come bundled with a CDROM or DVD containing about a thousand quiz questions. Do take the time to take all of the questions bundled with your book. Attempt the questions after you have finish reading each of the domains. This way you can gauge how much you have retained on each of the domains.

The quizzes at the end of the book will give you two advantages. The first one being the identification of your weak areas and the second advantage is that it will help you remember key topics within the CBK. It has been proven through scientific studies that quizzes are the best tool that you can use on top of reading ALL of the chapters within the books. Students who performed a large number of quizzes always perform better on the real exam.

The following are key domains you must master:

1. Information Security Governance and Risk Management
2. Access Control
3. Security Architecture and Design
4. Telecommunication and network security
5. BCP and DRP

You have probably heard that some of the domains are more important than others as far as the exam is concerned, this is true. However when you get a score of 698 and you miss passing the exam by one questions, let me tell you that you will regret it if you did not read and study ALL of the domains in the book.

Do read all of domains without exception. People that taught they were really good on some of the topics often time had the surprise of failing the exam because they knew too much and they were reading in between the lines too much. Reading and doing Quizzes will help you get the right mindset for the exam, you have to think like a manager and you have to think the way ISC2 wants you to think.

WHAT IS YOUR RECOMMENDED BOOKS?

Choosing a book is a bit like choosing a pair of shoes. Each person has its preference and it is hard to please everyone. A book has to be selected according to your taste and how much you already know about the 10 domains of the CISSP. Below you have my short list of recommended books:

The official **(ISC)2® Guide to the CISSP® CBK®, Second Edition** is the best book to find out what topics might be on the exam. If you are going to buy only one book that would be my recommended choice.

Recognized as one of the best tools available for the information security professional and especially for candidates studying for the (ISC)2 CISSP examination, the **Official (ISC)2® Guide to the CISSP® CBK®, Second Edition** has been updated and revised to reflect the latest developments in this ever-changing field. Endorsed by the (ISC)2®, this book provides unrivaled preparation for the certification exam that is both up to date and authoritative.

You can see other books I recommend at:

http://www.cccure.org/modules.php?name=News&new_topic=76

I am not a dummy should I buy the dummies book?

"The dummies book is a nice surprise. It is filled with tips and tricks and it is an easy read. I would not recommend it as you sole source but it is a great book for people who have years of experience or anyone who wish to perform a quick final review. It is a book I highly recommend in your final steps of preparation."

I HAVE MY BOOK, WHAT IS NEXT?

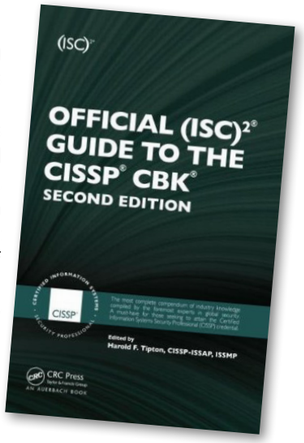
Buying books is the easy part, reading through and understanding the content is the hard part.

Do take the time to read ALL of the chapters carefully use a highlighter for key points and for identification of areas you had difficulties. Those points can be further discussed with your instructor when you take your boot camp or make use of the CCCure forums if you are not attending live training.

Regardless of your study path, I strongly recommend that you visit the CISSP Forum on the CCCure.Org portal. The forums are extremely lively and there are dozens of CISSP's in good standing that are waiting to help you and answer all of your queries.

You will find the forums at:

http://www.cccure.org/forum-3.html •



Clement Dupuis is the Chief Learning Officer (CLO) of SecureNinja.com. He is also the founder and owner of the CCCure family of portals.



For more information, please visit *http://www.cccure.org* or e-mail me at *clement@insyte.us*

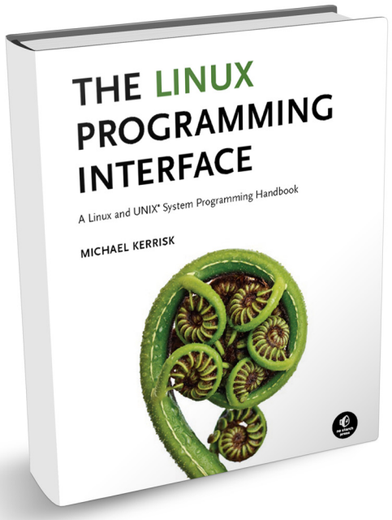
The CCCure Family of Portals:
http://www.cccure.org
For the CISSP in becoming and other high level certifications

http://www.freepracticetests.org/quiz/home.php
The CCCure FREE quizzer engine (25% of questions are FREE
We have 1800 questions for the CISSP EXAM



The Linux Programming Interface:
Linux and UNIX System Programming Handbook

by **Michael Kerrisk**



Edition: **1st, 2010**
Author: **Michael Kerrisk**
Publisher: **No Starch Press**
Pages: **1552, Hardcover**
ISBN-10: **9781593272203**

The Linux Programming Interface (TLPI) is the definitive guide to the Linux and UNIX programming interface—the interface employed by nearly every application that runs on a Linux or UNIX system.

In this authoritative work, Linux programming expert Michael Kerrisk provides detailed descriptions of the system calls and library functions that you need in order to master the craft of system programming, and accompanies his explanations with clear, complete example programs.

You'll find descriptions of over 500 system calls and library functions, and more than 200 example programs, 88 tables, and 115 diagrams. You'll learn how to:

- Read and write files efficiently
- Use signals, clocks, and timers
- Create processes and execute programs
- Write secure programs
- Write multithreaded programs using POSIX threads
- Build and use shared libraries
- Perform interprocess communication using pipes, message queues, shared memory, and semaphores
- Write network applications with the sockets API

While *The Linux Programming Interface* covers a wealth of Linux-specific features, including *epoll*, *inotify*, and the */proc* file system, its emphasis on UNIX standards (POSIX.1-2001/SUSv3 and POSIX.1-2008/SUSv4) makes it equally valuable to programmers working on other UNIX platforms.

T*he Linux Programming Interface* is the most comprehensive single-volume work on the Linux and UNIX programming interface, and a book that's destined to become a new classic.

About the Author
Michael Kerrisk has been using and programming UNIX systems for more than 20 years, and has taught many week-long courses on UNIX system programming. Since 2004, he has maintained the *man-pages* project (*http://www.kernel.org/doc/man-pages/*), which produces the manual pages describing the Linux kernel and *glibc* programming APIs. He has written or co-written more than 250 of the manual pages and is actively involved in the testing and design review of new Linux kernel-userspace interfaces. Michael lives with his family in Munich, Germany.

RATING ★★★★★

DTrace:
Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD (Oracle Solaris Series)

by **Brendan Gregg & Jim Mauro**

The first guide to DTrace: the breakthrough debugging tool for Mac OS X, Unix, Solaris, and OpenSolaris operating systems and applications

- Complete coverage: architecture, implementation, components, usage, and much more
- Covers integrating DTrace into open source code, and integrating probes into application software
- Includes full chapter of advanced tips and techniques
- For users of DTrace on all platforms
- Foreword by Bryan Cantril, creator of DTrace

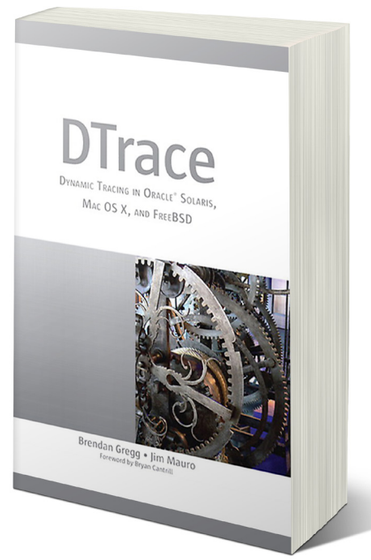
DTrace represents a revolution in debugging. Using it, administrators, developers, and service personnel can dynamically instrument operating systems and applications to quickly ask and answer virtually any question about how their operating systems or user programs are behaving. Now available for Solaris 10 and OpenSolaris, Mac OS X, and FreeBSD, thousands of professionals are discovering DTrace - but, until now, there's been no comprehensive, authoritative guide to using it. This book fills that gap. Written by four key contributors to the DTrace community, it's the first single source reference to this powerful new technology. The authors cover everything technical professionals need to know to succeed with DTrace, regardless of the operating system or application they want to instrument. The book also includes a full chapter of advanced tips and techniques.

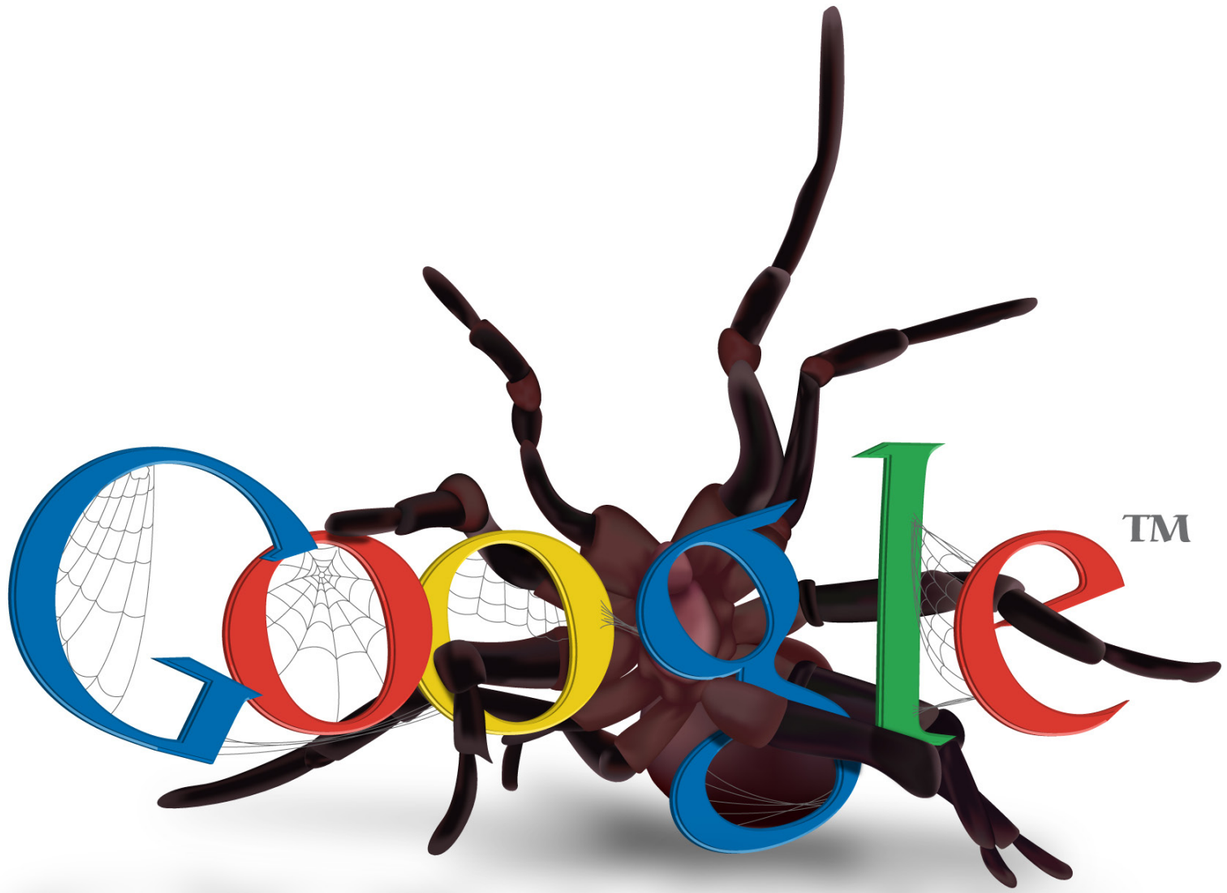
About the Author
Brendan Gregg, Staff Engineer at Sun Microsystems, works in the Fishworks engineering group alongside DTrace's creators. He created DTraceToolkit and DTrace FAQ, and co-authored several articles about DTrace.

Jim Mauro, Principal Engineer at Sun Microsystems, co-authored Solaris Internals.

RATING ★★★★

Edition: **1st, 2011**
Author: **Brendan Gregg & Jim Mauro**
Publisher: **Prentice Hall**
Pages: **1152, Paperback**
ISBN-10: **0132091518**





Vulnerability Reward Program

In line with the 'Economics of Vulnerabilities' keynote panel discussion at HITB2011 Amsterdam, we sit down with **Chris Evans** (Chrome Security) and **Adam Mein** (Security Program Manager) from Google Security Team to talk about Google's vulnerability rewards program.

Read on as they take us through the lessons they've learned, the problems they've encountered and how they actually decide what bugs are worth **$3133.7** and which are only **$1337**.

Is this a sign of things to come? Will 2011 be the year we see even more vendors jump on the bug bounty bandwagon?

Is the idea to expand the program to cover other Google web applications based on the success of the Chrome rewards program?

Adam Mein (AM): Very much so. From our experiences with the Chromium program, we knew we'd get more bugs, strong relationships and good value for money. I think this is a good method -- start with a single application and then use this experience to grow a bigger program.

Chris Evans (CE): Yes, I'm delighted with the success of the Chromium program. I'd also add that the Google Web program can already be declared a success, despite the short timeframe. We've paid out almost $200,000 of rewards and seen some really interesting bugs.

On your blog, you stated that the rewards program is "experimental". Does that mean this program could come to an end soon?

CE: Realistically, I don't see the program coming to an end. It's working too well to shut it down.

AM: Although it's unlikely, it's possible the proportion of low quality to high quality bugs will reach a point where we'd consider stopping the program. Since there's effort in triaging each bug and responding to the bug reporter, it's not a zero cost initiative. However, as Chris mentioned, I don't see us shutting them down anytime soon -- we're getting really good value at the moment.

Finding bugs in applications like Chrome can take weeks if not months. So why would a researcher choose your program when other security firms are known to pay at least double the amount currently offered by Google?

CE: This is an interesting question, and the answer comes down to an individual's primary motivations. I can offer two primary motivations that might lead a researcher to choose the Chromium Security Rewards program:

1) The researcher's primary motivation is keeping users safe. In this instance, filing the bug directly in the Chromium bug tracker (*http://crbug.com*) will get the bug to us fastest, and we'll fix it fastest. Sending the bug to a third party can introduce weeks of additional latency before we get a chance to fix it. In that time, a bad actor could rediscover the same bug and harm people with it. There's also the question of information sharing -- if you send your vulnerability information to a third party, who do they share it with? Does the government get sent a copy and if so, what do they do with it?

2) The researcher's primary motivation is to work with the

Chromium open source project. A lot of our contributors are open source fans, and users of Chromium or Chrome. These contributors enjoy working on finding bugs in the Chromium code base and generally giving back to open source. I really enjoy that we can send the occasional check to these contributors as a "thank you".

What about the black market trading of exploits? We've been told exploit writers can sell their wares for upwards of USD100,000 (ignoring all legal and ethical considerations)

CE: I'm not sure what to say other than, don't go there? Hopefully, we all got into security because we want to make things better for people.

I will add that we absolutely do not require a working exploit for bugs submitted to the Google programs. Taking Chromium as an example, simple evidence of memory corruption will get you considered for reward. Given that going from memory corruption to a reliable exploit can take weeks or even months, I recommend that people stop there and cash in at *http://crbug.com*.



AM: From a web perspective, I'm unsure whether a significant black market actually exists. It's not a great comparison, but if you chat with the bug brokers (ZDI, et al), web vulnerabilities are not currently a big part of their business, though I'm informed it is something that's of increasing importance. To reinforce what Chris said, we're not trying to compete with the black market - many of the people that report bugs are also heavy users of our services - they're keen to see bugs get fixed as quickly as possible. Getting a reward is the cherry on top.

How do you determine how much a researcher should be rewarded for a bug?

CE: For the Chromium program, there are four factors involved: the severity of the bug, the quality of the bug report, whether the bug is "clever" or unusual, and community involvement. Taking all of these into account, we come up with a figure that is usually $500, $1000, $1337, $3133.7 or some multiple or combination of these.

I get the most enjoyment out of rewarding $1337. It's not the highest level (which is reserved for Critical issues), but this level is usually reserved for a bug that particularly impresses the panel by being clever, devious or unusual. It's just a number, but it tells the researcher "you rock!".

AM: The web program is virtually identical, though we have the additional challenge that we're dealing with bugs in hundreds of different products - not just one - so the business impact for each vulnerability is also considered. Mostly, we don't differentiate between our different products - an XSS in YouTube is going to be worth the same as Google Docs, with a few exceptions for services such as Google Health, Gmail and Checkout. At the top end of scale, the bugs that get the greatest reward are generally the ones that impact many users in a really severe way. If you found a remote code execution or SQL injection bug that exposed a whole bunch of user info, this would be a candidate for top dollar.

Hopefully, we all got into security because we want to make things better for people

Who makes the final decision on the final reward amount?

CE: Both the Chromium Security Rewards program and the Google Web program have a panel of experts (these are named on the respective blog posts). This panel usually forms a consensus on each bug pretty easily.

On average, how much does each researcher get paid for each bug that they find and submit?

CE: For Chromium, most of our bugs are memory safety issues that manifest within the confines of the sandbox. For a good quality bug report for such a bug, we consistently reward at the $1000 level. You can look at our Hall of Fame: *http://www.chromium.org/Home/chromium-security/hall-of-fame*. As can be seen, $1000 is a very common reward amount.

AM: For the web program, the most common is $500, though we see a decent number of $1000 rewards.

Do you guys offer bonus rewards for those who are superstar 'exploiters'?

CE: Not yet, aside from the intangible benefits such as being considered for Google jobs and internships. I'm actually quite interested in providing motivations for 'fixing' as well as 'exploiting'. We've been increasing some of the rewards (up to a doubling) for people that approach us not only with a vulnerability but also a high-quality patch. I also wonder if I should be looking to provide extra motivations for new faces.

How many bugs have been reported and fixed in total?

CE: For Chromium, I'll give the link to the Hall of Fame again: *http://www.chromium.org/Home/chromium-security/hall-of-fame*. I try to keep it up to date. You can use it to count total number of bugs that were rewarded. It's something like 150 at the moment, and our total reward payout for the Chromium program is approaching $150,000. The Hall of Fame also lists some lower severity issues that didn't generate rewards, but it's still important to issue credit.

The majority of the bugs reported are in products and domains that aren't as widely used

AM: We don't list all the individual web bugs. In case people were curious for an approximate breakdown, I can answer that for you. The majority of the bugs reported are in products and domains that aren't as widely used. It's fairly unusual to get bugs reported in our most sensitive properties, such as Gmail, Checkout, Docs, etc., but we do encourage people to look.



How many bugs does Google receive on a daily basis?

CE: For Chromium, we probably get a few security bugs filed a day. Most of them are invalid (for example, not a security bug, or based on some misunderstanding).

Other than listing the contributors in the hall of fame page, are researchers allowed to make the vulnerabilities public once they have been patched?

CE: Yes, most definitely! Productively blogging and discussing our findings as a community is how we all advance our collective knowledge. So, it is not only allowed, it is encouraged. We have a pro-researcher culture. And why is that? It's because many of Google's security employees are themselves researchers in their personal time or sometimes even on company time. Also note that Chromium security bugs are opened to the public once they are fixed. That's currently a manual step so sometimes I get a little behind.

Why doesn't Google support this by officially making the vulnerabilities information public?

CE: For Chromium, we're an open source project so everything does become public. When I say everything, I mean everything. All the conversations we have with researchers are chronicled in the relevant security bug, and this bug becomes publicly viewable at some short time after we fix the bug.

AM: To be honest, most of the bugs are fairly boring. If it's an interesting bug, many of the top bug reporters choose to write up the details on their blogs -- we're highly supportive of this. It's particularly exciting to see this happen when there is something unusual about the bug that we can all learn from. If you're interested to find out more information, someone started a Twitter group made up of people who have received received rewards from us: *http://twitter.com/minetosh/halloffame* -- they will often post a link when they've written up the details of a bug.

Do you think that by making the vulnerability information public, more researchers would be encouraged to find similar bugs?

CE: As per above, Chromium bugs do become public. And, talking to some researchers, they do already read previous bugs and look at the code changes for those security bugs, in order to get ideas about where to look next.

OWASP listed "Open Redirection" as one of the top vulnerabilities for 2010, but Google does not consider this as a rewardable bug. Can you please elaborate more on why this is so?

CE: We only reward bugs above a certain severity. There's a still lot of debate amongst the security community on whether "open redirection" represents a security problem at all or not. Regardless of the outcome of that, I'm sure that no seasoned security professional would call them "serious".

I wrote a piece on my personal blog last year about this whole topic, explaining why people misunderstand open redirectors: *http://goo.gl/G7MuB*. The irony is that you just followed a link that is effectively an open redirector in order to read my blog post. And the point is that you can't tell

where you will end up by looking at where you're clicking. You need to pay attention to the browser's URL bar of the destination page in order to make trust decisions.

How long does it normally take for a researcher to get paid? I.e. from the time of submission to actual cash out?

CE: For Chromium, I start the pay-out process once we've released the fix to users. The pay-out process can take a little longer than expected because it turns out that "electronic" transfers are still slow in 2011! We have a bit of a reputation for fixing security bugs and releasing the fixes very quickly. To quote an example: *http://code.google.com/p/chromium/issues/detail?id=55350*. A nice privately-reported bug from security researcher Stefano Di Paola. We actually had a fix shipped to end users in about five days. We can't always guarantee to be that fast, but hopefully it shows that we take the responsibility of fast fixes seriously, and researchers shouldn't have to wait too long to get paid.

To be honest, most of the bugs are fairly boring



AM: The speed of payment is something that I'd like to improve for our web reward program. I don't know the exact figures, but it's rarely quicker than 3 weeks and often closer to 5. Some of the vulnerability reporters choose to batch up their payments and get paid in one large chunk. We're a little more flexible than Chris in terms of paying people - we commence the payment process when the bug is fixed OR two weeks after they've reported it - whichever comes first.

Have you guys recruited any of the researchers that participated in your program?

CE: No success stories yet, but I'm eagerly working on a few cases! Interestingly, many of our participants seem to be students. The future may well hold internships :)

Would you encourage other vendors to come up with their own rewards program? How would you advise them to get started?

CE: For us, it's been an overwhelmingly positive experience, so yes, I'd encourage other vendors to get on board. It'll probably be easier if you're a larger vendor, so that you have the install base and brand recognition that will attract researchers. We've seen a few smaller vendors attempt to start programs (possibly genuinely, possibly as a PR stunt), and these don't seem to have attracted the participants.

Getting started can be tricky. As a company, you need to have a lot of things in good order. For a start, you need products that aren't riddled with bugs. You need to care enough about remaining security bugs to fix them promptly. You need to be good at communicating openly, honestly and regularly with security researchers. You need to have a security team staffed up to accommodate a spike in load. A lot of larger companies who have products depended upon by millions fall down on some or even all of these things, so it's a shame that these things hold up their ability to start a rewards program.

One good idea is to start a program for a subset of your product portfolio. That enables you to start small and check you can handle the load before expanding the scope of the program. •

ADAM MEIN *Prior to joining Google in 2010, Adam worked for the Department of Defence in Australia. His background covers many of the typical IT security functions: policy formulation to incident response; penetration testing to education and training. Since starting at Google he's been focused on trying all the different snacks, getting better at pool and managing the program for externally reported vulnerabilities.*

CHRIS EVANS *Chris Evans is known for various work in the security community. Most notably, he is the author of vsftpd and a vulnerability researcher. Details of vsftpd are at http://vsftpd.beasts.org/. His work includes vulnerabilities in all the major browsers (Firefox, Safari, Internet Explorer, Opera, Chrome); the Linux and OpenBSD kernels; Sun's JDK; and lots of open source packages. He blogs about some of his work at http://scarybeastsecurity.blogspot.com/. At Google, Chris currently leads security for Google Chrome. He has presented at various conferences (PacSec, HiTB Dubai, HiTB Malaysia, BlackHat Europe, HiTB Amsterdam, OWASP, etc.) and is on the HiTB and WOOT paper selection panels.*



hitb magazine
KEEPING KNOWLEDGE FREE

HITB Magazine is currently seeking submissions for our next issue. If you have something interesting to write, please drop us an email at: *editorial@hackinthebox.org*

Submissions for issue #7 due no later than 23rd July 2011

Topics of interest include, but are not limited to the following:

* Next generation attacks and exploits
* Apple / OS X security vulnerabilities
* SS7/Backbone telephony networks
* VoIP security
* Data Recovery, Forensics and Incident Response
* HSDPA / CDMA Security / WIMAX Security
* Network Protocol and Analysis
* Smart Card and Physical Security
* WLAN, GPS, HAM Radio, Satellite, RFID and Bluetooth Security
* Analysis of malicious code
* Applications of cryptographic techniques
* Analysis of attacks against networks and machines
* File system security
* Side Channel Analysis of Hardware Devices
* Cloud Security & Exploit Analysis



Please Note: We do not accept product or vendor related pitches. If your article involves an advertisement for a new product or service your company is offering, please do not submit.



CONTACT US

HITB Magazine
Hack in The Box (M) Sdn. Bhd.
Suite 26.3, Level 26, Menara IMC,
No. 8 Jalan Sultan Ismail,
50250 Kuala Lumpur,
Malaysia

Tel: +603-20394724
Fax: +603-20318359
Email: *media@hackinthebox.org*